Towards A Data Privacy-Aware Execution Zone Creation on Cloud/Fog Platform

Somnath Mazumdar
Department of Digitalization
Copenhagen Business School
Solbjerg Plads 3, 2000 Frederiksberg, Denmark
sma.digi@cbs.dk

Thomas Dreibholz
Centre for Resilient Networks and Applications
Simula Metropolitan
Pilestredet 52, 0167 Oslo, Norway
dreibh@simula.no

***Abstract*—Cloud computing is now a 'go-to' platform for running various types of application. A wide spectrum of users needing software and hardware resources have embraced the cloud. Following the cloud business model, one is either a cloud-based service provider or a cloud service (hardware and software) user. There is a continuous evolution in the cloud ecosystem to support the ever-changing features of user applications. The cloud ecosystem has added a new resource delegation model to accommodate such new requirements. This new resource delegation model is known as fog. Both, cloud and fog platforms, employ complex hardware and software to provide better run-time support for applications. This paper presents a message-based privacy-aware application execution zone management framework for the cloud/fog platform. The framework aims to create a static execution zone based on the performance of the user application and the data privacy requirements. It also enables the user to set the resource termination conditions. Here, we have prototyped the proposed framework and showed how the proposed approach works with message structure and experiments with a P4 switch. We also present how the prototype could be implemented on a cloud/fog platform.**

***Index Terms*—Cloud, Data, Fog, Message, Privacy, Resource Management.**

I. Introduction

Cloud computing is now a well-accepted platform for various types of applications. However, the cloud is better suited for latency-tolerant applications and compatible with applications that can tolerate delays up to 100 ms [1]. Fog[1] has been developed to offer lower latency and is based on distributed heterogeneous platforms [2]. Fog units are placed between the cloud and the user, while being resourcefully supported by the cloud. The fog platform is composed of potentially resource-constrained devices [3], and managing such platforms is expensive and not trivial. Offloading user applications to fog can lead to performance improvements, and these applications may contain sensitive data and code. Third parties, including Internet service providers (ISPs), share online user data (such as real-time location) for their financial benefits [4]. Therefore, the preservation of the privacy of code and data is essential.

Problem Context. The recently coined term 'confidential computing' aims to create a space to process workloads securely [5]. Unfortunately, the core functionality of confidential computing is mainly dependent on the underlying hardware. Generally, standard virtual machines (VMs) hosted on hypervisors share memory to communicate with hosts and other VMs. Popular vendor-specific confidential computing-based memory solutions have also been shown to suffer from multiple issues, such as memory encryption, memory integrity, and attestation issues [6]. In general, existing solutions are code-focused, while users have to subscribe to other data management solutions for data security. Currently, there is no such solution where users of the public cloud service, primarily on the infrastructure-as-a-service (IaaS) layer, can get resources based on their specified application requirements *together with* data transmission restrictions.

This paper aims to fill the gap by prototyping a privacy-aware application deployment framework for a cloud/fog platform owner. It creates an 'execution zone' based on user-specified resources required for application and data transmission-related preferences. One such preference is user data privacy[2] and also preferred resource selection. User data sets can be spread across multiple clouds, including private data stores or private clouds. Our framework accepts user data with privacy preferences and other requirements as input. It sends the list of resources, such as VMs details, to users after applying the preferences. However, while the framework collects user preferences, authentication and authorisation must be enforced for increased data privacy. Overall, this work focuses on giving users more control over their data and allowing the service provider to automate the application deployment to a certain extent. Data transmission-related controls can be done by restricting countries, hardware, and routing.

Research Context. In this paper, we answer the research question: *How to generate an execution zone for a user-defined data privacy-aware application on a cloud/fog platform?* We assume that a single cloud service provider owns cloud and fog. Such a service provider collects user preferences to create an execution zone. These zones are based on application requirements and user data privacy policies. To do that, the platform owner will generate relevant messages. After the

[1]In this paper, we follow NIST's Fog definition.

[2]We define privacy as free from intrusion and able to control one's data, while security refers to data protection against unauthorised access to user data. In some cases, privacy and security may overlap.

successful creation of zones, users will receive the details of the resources to use for application deployment. If a user subscribes to one particular public cloud service provider, the fog service will only be from that provider. This step has been taken to reduce the inter-domain management issues, mainly the data transmission cost. The overall execution zone creation process is application-agnostic.

Data transmission management helps to restrict data in a particular region and route data traffic through the allowed path. In this way, we support user data privacy rules. Resources can be released via a control message. To realise the allocation and deallocation of resources, we developed two control messages for the platform owner. This control message-based approach can also support dynamic resource requirements to some extent, leading to better resource adaptation.

Generally, resource orchestration optimises existing computational resources with dynamic resource requirements while achieving service-level objectives. There are many scholarly resource orchestration-related works on fog [7] and cloud [8]. However, it is still an important problem. The proposed framework is based on centralised orchestration[3] control topology to reduce data communication issues. Generally, for distributed and decentralised orchestration control topologies, data exchange is required to implement the framework effectively [9]. Overall, in this paper, our contributions are:

1) We have proposed a message-based static resource allocation framework to create an application-agnostic privacy-aware execution zone. The primary aim of this paper is to propose and prototype such a framework.
2) There are two types of control messages: one is to set the resource creation and data transmission rules (termed as `INIT` message), and another is to release them (termed as `STOP` message). We also presented the message structures and implemented them.
3) We have shown how data transmission can be controlled using the encoded rules. We also discuss how this prototype can be implemented in a real setup.

Due to the delimitation of this research, no explicit data failure or data compression support is provided.

II. Related Work

Resource orchestration aims to offer the best viable assignment of available resources to meet the quality of service (QoS) and optimise execution time for the services while reducing energy costs. We refer the readers to the survey [7] that focuses on orchestration in fog, [8] for cloud platform, and [10] to learn more about the computation offloading process in the cloud ecosystem.

[11] presents a service provisioning mechanism for resource-competing cloud applications. The proposed mechanism creates execution zones to assign resources. [12] introduces a resource orchestration algorithm for optimal partitioning of shared cloud/edge resources. Here, each application can define its orchestration strategy through declarative statements. To find a trade-off between the platform owner's and the application's requirements, another workload orchestration framework is proposed in [13].

To preserve the privacy of smart home owners, a cloud/fog-based framework has been proposed in [14], while [15] manages dynamic IoT data processing flows for a cloud-native platform. Next, [16] presents a fuzzy decision tree-based task orchestration algorithm to support dynamical changes and heterogeneous devices. Overall, we found a gap in the literature, which focuses on the message-based creation of privacy-aware application execution zones, focusing primarily on data privacy for cloud-native platforms.

[3]In this paper, we define 'orchestration' as 'an effective resource arrangement to generate an application-specific execution zone'.

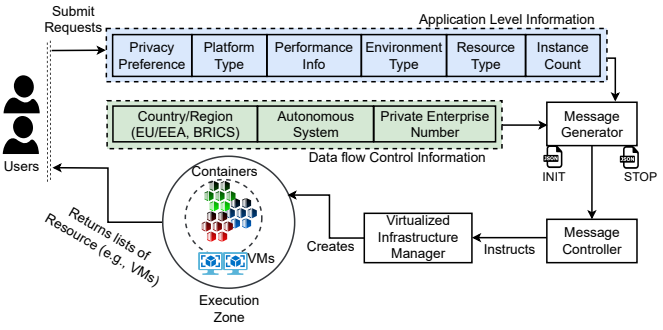

Figure 1: Application and data-level message creation process of the proposed framework.

III. Proposed Framework

The proposed message-based framework (see Figure 1) is aimed at the owner of a cloud/fog platform that offers services at the IaaS level. In such a setup, a subscriber/user will request cloud/fog resources in the form of VMs and containers with their required quantity. The proposed framework allows users to add preferences related to the code execution environment and data transmission. The resources will be statically mapped based on application-level information (blue shade of Figure 1), and the resources that will exchange data may follow the user preferences (green shade of Figure 1). Users can run any type of application. If the data packet transmission during execution does not meet the rules, the packets will be discarded. After satisfying the user preferences, he/she will receive a list of VM or container instances (including their IP addresses) to deploy applications. It is worth noting that the framework uses a first-fit matching method to satisfy user resource preferences. If there is no match, the user will be informed. Next, the user could decide to change priorities and try again or wait. The framework does not use any complex resource allocation algorithm, but such an algorithm can be added.

A. Primary Stages

The proposed techniques are based on three primary stages. These steps are:

1) *Receiving Request:* Resource usage requests can be submitted by a RESTful API.
2) *Request Handling:* After receiving application-specific requirements, the service provider verifies whether the fog has available resources. If not, the user will be informed to change preferences or wait until the resources are released. It also checks data privacy preferences simultaneously.
3) *Resource Allocation:* Broadly, resource orchestration can be divided into resource mapping, scheduling, and data placement. The proposed framework only supports *static* resource mapping and privacy-aware data placement. This stage outputs the VM/container IP details, constituting an 'execution zone' for a requested application.

The scenarios of computing resource consumption of applications can change dynamically. In general, adding or removing VMs or container instances according to load (known as horizontal scaling) or resizing existing allocated resources (known as vertical scaling) is popular. Our aim is to make the process simple and easy to implement. To achieve this goal, the framework has two primary features. They are *i)* customised execution zone creation satisfying user-specified data transmission constraints (refer to Subsection IV-A); and *ii)* releasing computing resources (refer to Subsection IV-E) after execution.

B. User-Level Information Collection

1) Application Level: The user can provide six types of application-level information during the resource request process (see the blue shade of Figure 1). First, the user specifies whether or not the application's privacy should be preserved. If so, then the preferences related to data flow must be mentioned (see Subsection III-B2). Next, the user must specify whether cloud and fog will be used. The user mentions whether the application is latency-sensitive or latency-tolerant (denoted as performance info). The user also needs to provide details related to the type of environment. The application can run in a container (fog) or a VM (mainly in the cloud). The fog will be the default choice if the application is latency sensitive. If required resources are unavailable, the user must wait until the necessary resources become available. Users must also declare the required resource type within the preferred platform. Some hardware is only available in the cloud, such as FPGAs[4] and large GPUs[5]. This communication between the platform and the user is done via messages using JavaScript Object Notation (JSON) format. As it is a static mapping among the resources, the user is responsible for scheduling applications on the received resources (i.e. VMs and containers) and handling run-time management issues.

2) Data Level: The framework collects three primary pieces of information (see green shade of Figure 1) for providing privacy-aware data transfer. They are *i)* country, *ii)* autonomous system (AS), and *iii)* private enterprise number (PEN). These details must be encoded in the rules of the execution zone for data transmission. We have assumed that the number of data flow control messages will equal the number of user data sets. However, configuring per-data flow rules is also possible. Adopting a message-based framework allows us to simplify the control flow mechanism. Router logic enables data packet transfer. For instance, the data traffic can be confined to a geography/region level, ISP level, and network equipment level.

[4]FPGA: Field Programmable Gate Array.

[5]GPU: Graphics Processing Unit.

C. Information Processing and Resource Allocation

Upon receiving user input, the message generator creates an `INIT` message (refer to Figure 1). The `INIT` message contains application and data-level privacy specifications. The message controller uses this information to create an execution environment, including provisioning instances by the cloud/fog service provider via the Virtualised Infrastructure Manager (VIM). The proposed framework would create a separate virtual LAN (VLAN) for the application. Different VLANs shield unrelated execution environments from each other. It means that a security breach in one application will not affect other co-located applications. It also provides privacy-aware routing to all cloud/fog resources and routing to the Internet (if necessary).

In general, ISPs do not provide privacy-aware routing, while [14] suggests using virtual private networks (VPN) over non-privacy-aware ISPs. To prevent the derivation of privacy-relevant information from encrypted VPN user data traffic, [14] proposes a switch/router-based framework that uses such VPN connections to aggregate traffic from multiple IoT applications. Generally, there are two ways of specifying privacy preferences in IP packets:

1) Using the DiffServ Code Point (DSCP): This approach does not increase packet sizes, but limits marking capabilities to just 6 bits. Because only 6 bits of the IPv4 Type of Service (TOS) field or IPv6 Traffic Class field in the IP header are available for customisation. In this paper, we have selected this option.
2) Adding an IPv4 option or IPv6 extension header: This allows more flexibility with variable-length privacy information. However, adding additional information increases packet sizes. It means that it needs a larger maximum transmission unit (MTU). Alternatively, the MTU can remain unchanged, leading to a smaller payload (i.e. maximum segment size (MSS) of the Transport Layer protocol) or fragmented.

For implementation, we have used a Programming Protocol-independent Packet Processors (P4) [17] based switch to provide marking and packet processing. P4 is standard and provides an open, vendor-independent way to realise custom functionalities in network devices. It allows us to implement advanced features for marking and custom-protocol deep packet inspection. Network monitoring may be used for resource deletion to identify an idle execution environment. Such monitoring could be done using the proposed framework thanks to P4.

```
{
  "type":            "INIT",
  "application_id": "a94e7...82491",
  "privacy": {
    "regions":   [[ALLOW_SPECIFIC,"EEA"],
  ↪  [DENY_ALL]],
    "countries": [[DENY_SPECIFIC, 240]],
    "as":        [[DENY_SPECIFIC, 3320]],
    "equipment": [[DENY_SPECIFIC, 9]]
  },
  "performance":     "latency",
  "environment":     "container",
  "platform_type":   "cloud+fog",
  "resource_type":   "CPU",
  "instance-count": 2,
  "instance-details": {
    "cores":   2,
    "memory":  2048,
    "storage": 16384,
    "image":   "Ubuntu-22.04"
  }
}
```

Listing 1: JSON format of `INIT` message.

IV. Processing of Preferences

There are two types of messages the message generator (refer to Figure 1) can send to the controller: *i)* `INIT` message: to create the data privacy-aware application execution zone and *i)* `STOP` message: to release the unnecessarily held resources. In other words, an execution zone created by an `INIT` message can be deleted by a corresponding `STOP` message. A `STOP` message can be created by the user's prespecified conditions or the system's current state, such as memory or I/O stalls or CPU usage threshold.

The message controller generates both control messages, which the controller intercepts. As a next stage, it creates configuration instructions for VIMs (of clouds/fogs; i.e. create networks, instances), network and P4 switch(es).

A. Execution Zone Creation via `INIT` message

Each application uses a unique identifier in the form of a Universally Unique IDentifier (UUID), which is `application_id`. An example `INIT` message is presented in Listing 1, which contains both the privacy and the resource specifications (see Figure 1). It identifies the execution environment and leads to allocating a network address, including VLAN ID. It is worth noting that users can manually assign or request via the `INIT` message if a running application requires more resources.

B. Applying Data Privacy to Execution Zone

Privacy information can be found in the `Privacy Preference` block (refer to Figure 1). It contains constraints for:

1) `Countries`: The countries are supported by enlisting the ITU-T E.212 mobile country code[6] (MCC).
   - `Region(s)`: Specification of a geographical/economic/political region string. For instance, European Union (EU), European Economic Area (EEA)[7], and BRICS[8]. Each region is mapped to a list of countries. Regions are used to simplify the specifications.
2) `AS`: AS are denoted by ISPs and transit networks by the Internet Assigned Numbers Authority (IANA) Autonomous System (AS) number[9].
3) `Equipment`: Networking equipment vendors can also be listed by their IANA private enterprise number[10] (PEN).

C. Enforcing Data-Flow-Related Constraints for Execution Zone

The message controller ensures the data-flow-related constraints specified in the `INIT` message are handled effectively. Each constraint (total four) is specified by lists, where each item is a tuple of action (`DENY_SPECIFIC`, `DENY_ALL`, `ALLOW_SPECIFIC`, `ALLOW_ALL`) and value (i.e. `region/country/AS/PEN`). The lists of all four constraints are checked sequentially. According to the actions, the VIM matches the requirements if its properties do not lead to a denial.

- `DENY_SPECIFIC`=0: The specified value is prohibited. If it matches the cloud, it cannot be used.
- `DENY_ALL`=1: Any value is prohibited. There is no need to specify a value. `DENY_ALL` can be used for the last entry of the constraint list for disallowing all other values.
- `ALLOW_SPECIFIC`=2: The specified value is allowed.
- `ALLOW_ALL`=3: Any value is allowed. There is no need to specify a value.

The example in Listing 1 (refer to the disclaimer in Section VII) uses `ALLOW_SPECIFIC` to allow the 'EEA explicitly' `regions` (i.e. all EEA countries), but deny any other region with `DENY_ALL` as last tuple. We used constants (`DENY_SPECIFIC`, `DENY_ALL`, `ALLOW_SPECIFIC`, `ALLOW_ALL`) here for illustration, but the JSON message uses the numeric values above. The `countries` specification applies `DENY_SPECIFIC` to disallow MCC `240` (Sweden), allowing any other country with implicit `ALLOW_ALL` (no need to specify). Together with the `regions` setting, the example would allow all EEA countries except Sweden. Furthermore, the `as` constraint disallows using the network of Deutsche Telekom (AS 3320), while the `equipment` constraint disallows Cisco devices (PEN 9). The resource specifications are checked when a cloud matches the privacy requirements.

D. Applying Other Application Preferences

In addition to the `Privacy Preference` option, the user needs to submit five additional pieces of information. They are:

1) `Platform Type` specifies whether fog, cloud, or both resources are required during application deployment.

[6]MCC: https://en.wikipedia.org/wiki/Mobile_country_code.

[7]EEA: Covers the EU, Iceland, Liechtenstein, and Norway.

[8]BRICS: Brazil, Russia, India, China, and South Africa.

[9]AS: https://www.cidr-report.org/as2.0/autnums.html.

[10]PEN: https://www.iana.org/assignments/enterprise-numbers/enterprise-numbers.

```
{
  "type":           "STOP",
  "application_id": "a94e7...82491",
  "instances": {
    "Fog2": [ "C3" ]
  }
}
```

Listing 2: JSON format of `STOP` message.

2) `Performance Information` denotes the application type, either latency-sensitive or latency-tolerant.
3) `Environment Type` denotes that the VM, container, or both will be used during deployment.
4) `Resource Type`: Many recent applications (such as machine learning) require variable CPU core counts or special-purpose accelerators (such as FPGAs and GPUs).
5) `Instance Count`: When requesting the resource type, users also needed to list the required number/capacity of the hardware.

If a cloud is compatible with the requirements, the VIM (refer to Subsection IV-F) of the cloud/fog is contacted for the actual availability of resources according to the number of instances and the details of the instance (i.e., number of cores, amount of memory and storage, the requested VM/container image). When available, instances and the underlying network configuration are created. It also includes VLAN configuration and VPN connections. For an existing execution environment, using the `INIT` message can also support additional resource requirements (resource scaling).

E. Resource Release via `STOP` Message

Generally, in public cloud platforms, the users are responsible for releasing the resources after use. The `STOP` message can terminate stalled or completed applications without user intervention. Faster termination of unnecessarily held fog resources helps to accommodate more fog users. The `STOP` message aims to automate this process. Listing 2 represents the JSON structure of the `STOP` message, which contains the `application_id` (see Subsection IV-A). If there is no further specification of which instances are to be deleted, the whole execution environment is terminated. Otherwise, only the selected resource (here: container instance 'C3' in 'Fog2') is released.

A `STOP` message can help the user to enforce the rule of whether memory or I/O stalls are allowed. This situation is relevant when an application is stuck or fails due to buggy code. User-specific conditions, such as the execution of deadline-based applications, can also be used as another condition for resource termination. Furthermore, if a running application requires fewer resources, a `STOP` message can be used to release unnecessary resources.

F. Resource Management via VIM

A VIM generally manages the compute (CPUs, GPUs, FPGAs), storage, and physical and virtual networks, including ports and access control within the corresponding cloud/fog infrastructure. That is, it manages the virtualised infrastructure (including VM and container images) and provides system-state-level monitoring for performance engineering. It is a vital component of an orchestration architecture used at the IaaS level (refer to Figure 2).

OpenStack is a well-known VIM hosting VMs, while Kubernetes is popular for hosting containers. We can add another abstraction layer to manage the composition of virtual network functions (VNFs) to network services, which are instantiated via VIMs in the underlying virtualisation infrastructures. Open Source MANO (OSM)[11] is an example of such a management and orchestration software stack.

The proposed framework is independent of the VIM and its underlying virtualisation infrastructure. The message controller communicates with the VIM via the Control API (typically a RESTful API with JSON messages) to instantiate networks and VMs/containers and to remove them after use. It is only necessary to add infrastructure-dependent VIM communication by extending the message generator to support another public cloud provider.

Table I: Available cloud with platform level information.

Name, Country (Code)	Environment	Platform	Resources	Equipment
Fog1, NO (242)	Container	Cloud/Fog	C	HP (11), Dell (674)
Fog2, DK (238)	Container	Cloud/Fog	C, G	Huawei (2011), ZTE (3902)
PrivateCloud1, NO (242)	VM	Cloud	C, F	Huawei (2011)
PrivateCloud2, DK (238)	VM	Cloud	C, F	Dell (674), Fujitsu (79)
PublicCloud1, US (310)	VM	Cloud	C, G, F	Cisco (9), Dell (674)
PublicCloud2, RU (250)	Container	Cloud	C, G	Huawei (2011)
PublicCloud3, CN (460)	VM	Cloud	C, G, F	Huawei (2011), ZTE (3902)
PublicCloud4, DE (262)	VM	Cloud	C, F	Siemens (1894), Robotron (44957)

V. Evaluation

We have evaluated our proposed framework using a cloud/-fog research test bed. Our local private cloud setup comprises devices connected to a custom P4-based switch. `Port-0` of this switch is connected to a router, connecting the setup over the Internet. The cloud/fog ecosystem, including the P4 switch, runs on Ubuntu Linux. In our target switch, a P4 program realises the required functionalities, such as a MAC-to-port forwarding table, while using privacy rules.

A. Experimental Setup

Table I provides the available clouds/fog providers of the example, containing fog together with private and public clouds in China (CN), Denmark (DK), Germany (DE). Norway (NO), Russia (RU) and United States (US). They provide different types of environments (VM or container) with varying types of resources (i.e. CPU (denoted as 'C'), GPU (denoted as 'G'), and FPGA (denoted as 'F') in Table I). In our setup, we rely on VPNs. It means that we do not expect any privacy support from ISPs, which would be difficult in reality. We also assume that we establish our own VPN connections between the local network (Home) and the cloud and, therefore, do not specify AS numbers here. Instead, we only use example equipment PENs.

B. Experimental Results

[11]Open Source MANO (OSM): https://osm.etsi.org/docs/user-guide/latest/.

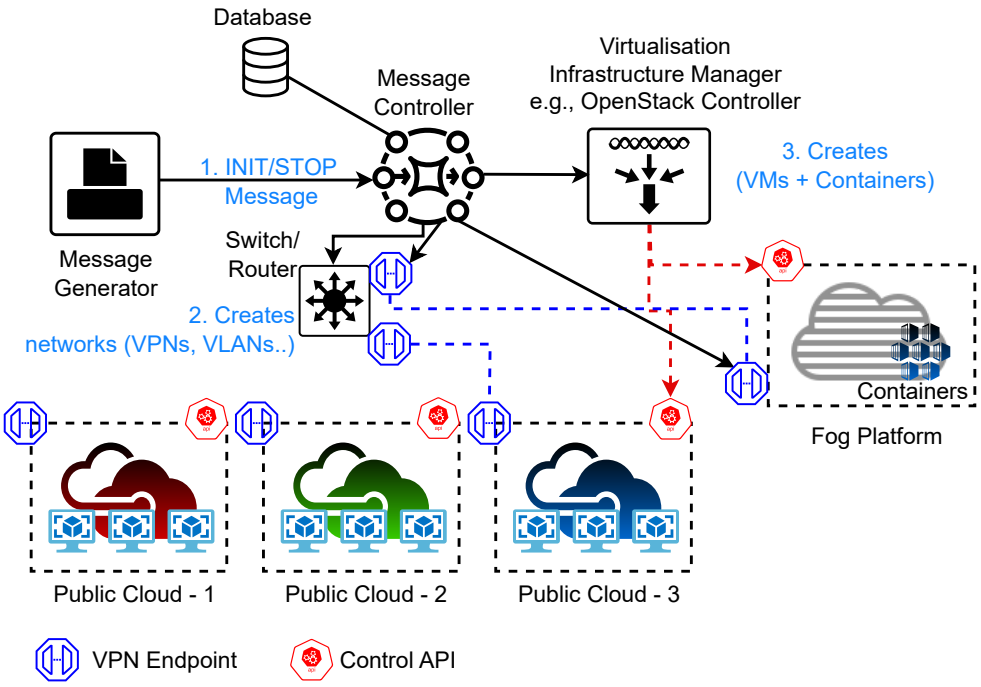

Figure 2: Overview of resource orchestration process during deployment at cloud/fog platform.

1) Control Message Functionality Checking: Eight example requests are summarised in Table II, with the relevant details from the corresponding `INIT` message (see Subsection III-C). To improve readability, we represent MCCs and PENs by names and use (✓,value) for `ALLOW_SPECIFIC` and (×,value)/(×) for `DENY_SPECIFIC`/`DENY_ALL` to declare privacy requirements (as explained in Subsection IV-A). In particular, requests have privacy requirements. For instance, EEA without Sweden (SE) for the first request (755b9...04f51), BRICS without South Africa (ZA) for the third (5050f...522ca), America without Canada (CA) for the fourth (83042...2aaed), or only allowing Siemens and Robotron equipment for the sixth (ded95...62d9e). The first six requests create new execution environments, while the last two *increase the resources* of existing execution environments (i.e. resource scaling). Furthermore, Table III shows two `STOP` messages, deleting all of the sixth (ded95...62d9e) and selected instances (i.e. scaling down) of the second execution environment (9ed1d...0eb38).

Listing 3 shows an excerpt of the execution environments created and configured in a JSON structure. It contains the execution environments with their `application_id`, VLAN allocation in the home network, IP network allocations, VPN settings, and the container or VM instances with their addresses. Note that our system uses IPv6 Unique Local addressing [18] to avoid global ambiguities. This is not strictly necessary from a functional perspective (the same private IPv4 address ranges may be reused in different VLANs). However, it helps to detect security-relevant configuration issues, such as misconfigured VLANs.

```
{
  "755b94a1-2847-4bb8-b530-cfca97404f51": {
    "vlan_id": 1000,
    "vpn": {
      "Fog1": {
        "from": "HOME-SWITCH",
        "to": "Fog1-SWITCH"
      },
      "Fog2": {
        "from": "HOME-SWITCH",
        "to": "Fog2-SWITCH"
      }
    },
    "network_id": 274468950876,
    "network": "fd3f:e79f:d35c:3e8::/64",
    "instances": {
      "Fog1": {
        "C1": {
          "cores": 2,
          "memory": 1024,
          "storage": 8192,
          "address": "fd81:50db:efe3:3e8::1:1"
          ...
        },
        ...
      },
      "Fog2": {
        ...
      }
    }
  },
  "9ed1d0da-14f0-4549-87de-6f447df0eb38": {
    ...
  },
  ...
}
```

Listing 3: Execution environments after processing all `INIT` and `STOP` messages listed in Table II.

Table II: Processed `INIT` message requests (NB: Exec. Env. means Execution Environment).

Exec. Env.	Constraint Type	Constraint	Value
755b9...04f51	Privacy	Regions	(✓,EEA), (×)
755b9...04f51	Privacy	Countries	(×, SE)
755b9...04f51	Resources	Performance	latency
755b9...04f51	Resources	Environment	container
755b9...04f51	Resources	Platform_type	cloud+fog
755b9...04f51	Resources	Resource_type	CPU
755b9...04f51	Resources	Instance_count	2
755b9...04f51	Resources	Cores	2
755b9...04f51	Resources	Memory	1024
755b9...04f51	Resources	Storage	8192
9ed1d...0eb38	Privacy	Regions	(✓,EU), (×)
9ed1d...0eb38	Privacy	Countries	(×, SE)
9ed1d...0eb38	Resources	Performance	latency
9ed1d...0eb38	Resources	Environment	VM
9ed1d...0eb38	Resources	Platform_type	cloud
9ed1d...0eb38	Resources	Resource_type	CPU
9ed1d...0eb38	Resources	Instance_count	3
9ed1d...0eb38	Resources	Cores	4
9ed1d...0eb38	Resources	Memory	2048
9ed1d...0eb38	Resources	Storage	16384
5050f...522ca	Privacy	Regions	(✓,BRICS), (×)
5050f...522ca	Privacy	Countries	(×, ZA)
5050f...522ca	Resources	Environment	container
5050f...522ca	Resources	Platform_type	cloud
5050f...522ca	Resources	Resource_type	CPU
5050f...522ca	Resources	Instance_count	8
5050f...522ca	Resources	Cores	2
5050f...522ca	Resources	Memory	1024
5050f...522ca	Resources	Storage	8192
83042...2aaed	Privacy	Regions	(✓,America), (×)
83042...2aaed	Privacy	Countries	(×, CA)
83042...2aaed	Resources	Environment	VM
83042...2aaed	Resources	Platform_type	cloud
83042...2aaed	Resources	Resource_type	GPU
83042...2aaed	Resources	Instance_count	4
83042...2aaed	Resources	Cores	16
83042...2aaed	Resources	Memory	8192
83042...2aaed	Resources	Storage	65536
f885e...89c29	Privacy	Regions	(✓,EEA), (×)
f885e...89c29	Privacy	equipment	(×,Huawei)
f885e...89c29	Resources	Environment	VM
f885e...89c29	Resources	Platform_type	cloud
f885e...89c29	Resources	Resource_type	CPU
f885e...89c29	Resources	Instance_count	8
f885e...89c29	Resources	Cores	4
f885e...89c29	Resources	Memory	2048
f885e...89c29	Resources	Storage	16384
ded95...62d9e	Privacy	Regions	(✓,EEA), (×)
ded95...62d9e	Privacy	equipment	(✓,Siemens), (✓,Robotron), (×)
ded95...62d9e	Resources	Environment	VM
ded95...62d9e	Resources	Platform_type	cloud
ded95...62d9e	Resources	Resource_type	FPGA
ded95...62d9e	Resources	Instance_count	3
ded95...62d9e	Resources	Cores	16
ded95...62d9e	Resources	Memory	8192
ded95...62d9e	Resources	Storage	65536
5050f...522ca	Privacy	Regions	(✓,BRICS), (×)
5050f...522ca	Privacy	Countries	(×, ZA)
5050f...522ca	Privacy	equipment	(×,Cisco)
5050f...522ca	Resources	Environment	VM
5050f...522ca	Resources	Platform_type	cloud
5050f...522ca	Resources	Resource_type	FPGA
5050f...522ca	Resources	Instance_count	2
5050f...522ca	Resources	Cores	16
5050f...522ca	Resources	Memory	8192
5050f...522ca	Resources	Storage	65536
755b9...04f51	Privacy	Regions	(✓,EEA), (×)
755b9...04f51	Privacy	Countries	(×, SE)
755b9...04f51	Resources	Performance	latency
755b9...04f51	Resources	Environment	container
755b9...04f51	Resources	Platform_type	cloud+fog
755b9...04f51	Resources	Resource_type	GPU
755b9...04f51	Resources	Instance_count	2
755b9...04f51	Resources	Cores	2
755b9...04f51	Resources	Memory	1024
755b9...04f51	Resources	Storage	8192

Table III: Processed `STOP` message requests.

Execution Environment	Instances
ded95...62d9e	(all)
9ed1d...0eb38	PrivateCloud2/V2

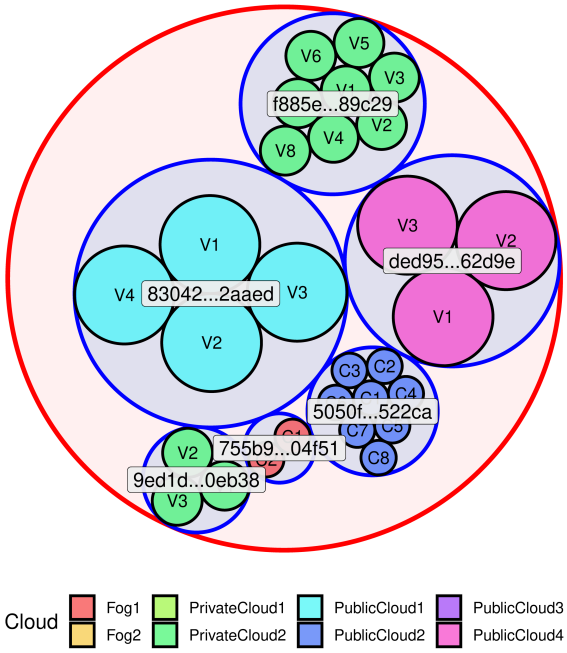

Figure 3: Execution environments after processing the first six `INIT` messages of Table II.

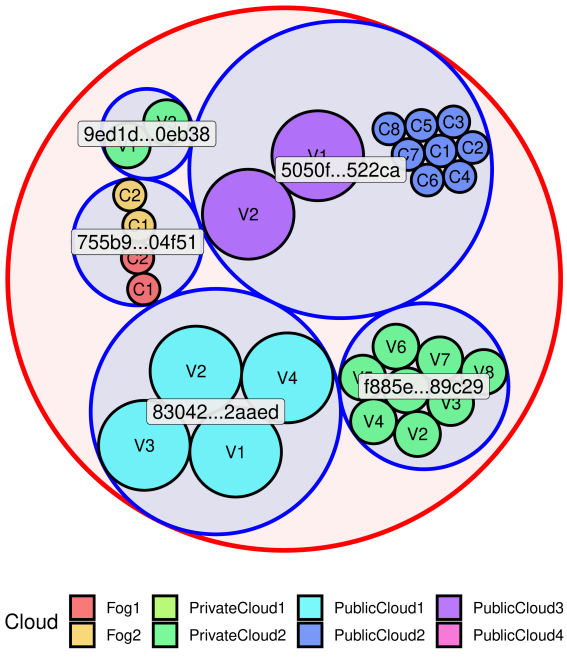

Figure 4: Execution environments after processing the all `INIT` and `STOP` messages of Table II and Table III.

2) Resource Visualisation: To make the resulting configuration more intuitive, we also visualised the output of Listing 3 in Figure 3 (after processing the first six `INIT` messages of Table II) and Figure 4 (after processing all `INIT` messages of Table II, and the `STOP` messages of Table III). The figures show the set of all execution environments (using the red line colour), where each execution environment is drawn as an outer circle (using the blue line colour). The inner circles (using black line colour) represent the enumerated VM (Vn) or container (Cn) instances at each cloud/fog provider, where the background colour corresponds to the clouds/fogs of Table I. The size of the instance circles corresponds to the number of cores.

During the creation of the six execution environments (i.e. the first six requests; refer to Figure 3), the first request (755b9...04f51) for cloud and fog resources in the EEA without Sweden is serviced by the fog provider 'Fog1' in Norway with two container instances. The second request (9ed1d...0eb38), asking for three cloud VMs in the EEA without Sweden, gets fulfilled by 'PrivateCloud2' in Denmark. The third request (5050f...522ca) for eight cloud containers in the BRICS region without South Africa is served by the Russian public cloud provider 'PublicCloud2'. The sixth request (ded95...62d9e) for three cloud VMs in the EEA with Siemens and Robotron equipment uses the German cloud provider 'PublicCloud4'.

After creation, the setup is scaled by two more `INIT` messages and two `STOP` messages (see Figure 4): The sixth execution environment (ded95...62d9e; with the three VMs in 'PublicCloud4') is completely deleted. The second execution environment (9ed1d...0eb38; with three cloud VMs in 'PrivateCloud2') is scaled down: VM instance 'V2' is deleted, leaving only the VM instances 'V1' and 'V3'. The third execution environment (5050f...522ca; with eight cloud containers in 'PublicCloud2') is scaled up with two VMs providing FPGA. Since the Russian 'PublicCloud2' does not provide FPGA, this adds two instances in the Chinese 'PublicCloud3'. It also leads to the addition of another VPN connection for the additional cloud provider. Finally, the last request scales up the cloud/fog resources of the first execution environment (755b9...04f51) by adding GPU resources. Since 'Fog1' only provides CPU resources, the requested GPU resources are fulfilled by adding 'Fog2' with two containers. They may need an additional VPN connection to 'Fog2'.

3) Privacy Markings Processing: A DSCP-based marking scheme is presented in Table IV for an additional VPN connection setup, defining bits in the Traffic Class/TOS field corresponding to the allowed regions. Note that regions may overlap, e.g. Scandinavia (Bit 7) is part of the EEA (Bit 6) and has some overlap with the EU (Bit 5). This scheme is simple but does not provide all the required constraints. Therefore, even when applying this simplified marking, additional knowledge is necessary to ensure conditions such as prohibited equipment or countries. Table V shows the corresponding simplified privacy markings as Traffic Class/TOS bits. The alternative and more complex solution would be to let the P4-switch add a header with full privacy restrictions as in Table II.

Table IV: Using Traffic Class/TOS for privacy marking.

Bit	Description
7	Allow Scandinavia (DK, NO, SE)
6	Allow EEA area (GDPR privacy rules)
5	Allow EU area (GDPR privacy rules)
4	Allow American area (Americas, i.e. US, CA, etc.)
3	Allow BRICS area (BR, RU, IN, CN, ZA)
2	Allow North Africa and Middle East

Table V: Traffic Class/TOS privacy marking in the VLANs.

Execution Env.	VLAN ID	7	6	5	4	3	2
755b9...04f51	1000	1	1	1	0	0	0
9ed1d...0eb38	1001	0	0	1	0	0	0
5050f...522ca	1002	0	0	0	0	1	0
83042...2aaed	1003	0	0	0	1	0	0
f885e...89c29	1004	1	1	1	0	0	0
ded95...62d9e	1005	1	1	1	0	0	0

Assuming that the ISP supports the marking schema, and 'Fog2' (in Denmark) could be reached via another provider supporting the privacy marking scheme, routing to 'Fog2' would be possible without VPN. However, for the first execution environment (755b9...04f51), it must be ensured that the routing does not involve Sweden (see the restrictions in Table II). If this cannot be ensured (which is likely), or the underlying networks do not support privacy marking (very likely), VPNs are necessary to ensure privacy. Therefore, Listing 3 also shows a VPN to 'Fog2'. Note that one VPN connection may be shared by different VLANs, combining (but not mixing, the VLANs would still be kept separately within a VPN connection) traffic to camouflage the applications' traffic patterns. [19] shows that multiple ISPs (including ISPs with different privacy-enabled interconnections) can also be supported.

4) Target Execution Environment: In this paper, the framework's primary aim is to show the functionality of two control messages while allocating the execution zones. Figure 5 illustrates how the execution environment could be set up for multiple users. The P4-switch has assigned a separate VLAN for each of the N execution environments. On `Port 0`, the switch connects a trunk port to a router. The router connects to the Internet. It can also maintain VPN connections. Depending on the implementation, both functionalities could be integrated into one device.

VI. Discussion

In the following, we will briefly discuss routing configurations, the applicability, and lessons learnt from this work.

A. A Note on Routing Configuration

We already mentioned in Subsection III-C that no free space is available in the IPv4 or IPv6 header, except for the six bits of the DSCP as part of the Traffic Class/TOS field. Adding an extra header will increase the overhead and also the packet

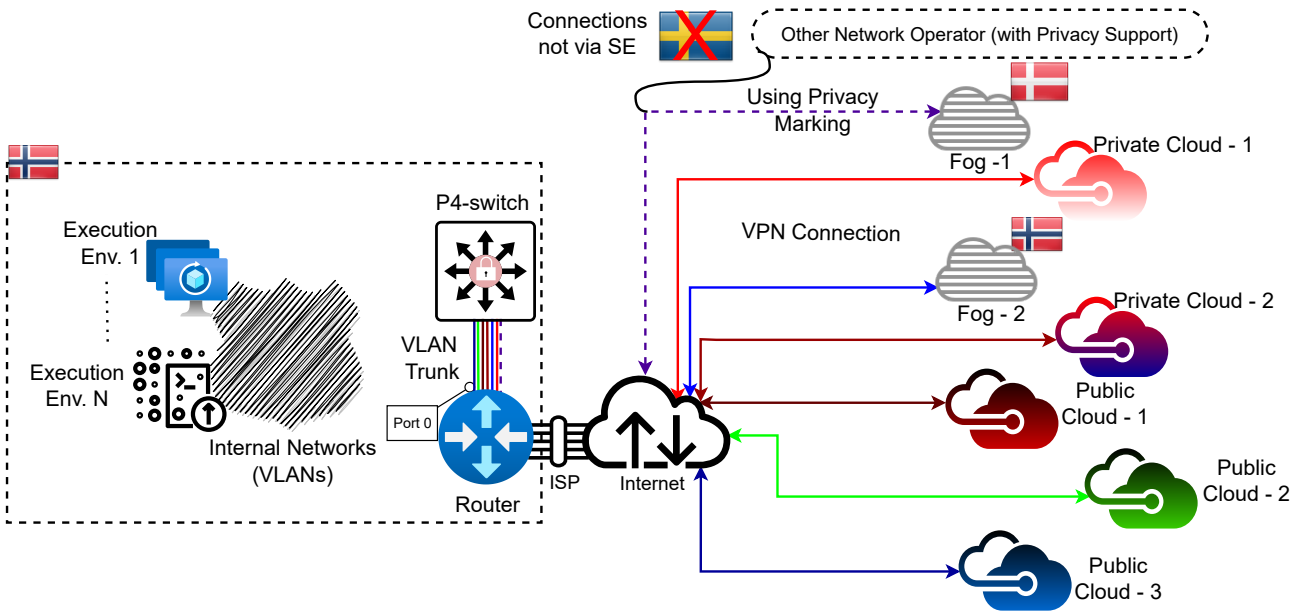

Figure 5: Possible implementation of user-specific multiple execution environments.

length. To avoid this problem, we have used the DSCP-based approach.

1) Source Routing: On another side, for security reasons, IP source routing is usually disabled. A privacy-focused source routing requires remote network topological information. Such details are only available to ISPs. They are also security-relevant. That means that it is unlikely that an ISP will reveal such information. Furthermore, source routing would provide a possibility for network-based attacks [20]. Therefore, it is not practically applicable.

2) MPLS: Existing Multi-Protocol Label Switching (MPLS) is expensive but can be ideal not only for real-time applications. MPLS lacks encryption and cannot easily set up a direct network connection to specific cloud servers. Overall, MPLS sends packets along predetermined network paths and does not support a series of intermediary destinations. However, correctly choosing the predetermined network paths makes it possible to fulfil paths and hardware constraints while keeping the latency low.

3) VPN: Generally, VPNs encrypt user data packets. Users use VPNs to protect against snooping. While a VPN protects the packet contents, it still allows one to derive some privacy-related information from the encrypted packets. For instance, the data volumes, packet sizes, and interarrival times let one assume the type of application and usage pattern (e.g. video streaming, voice call, and bulk data download). Ideally, data from several applications, with added dummy traffic (i.e. additional traffic, causing additional cost) and traffic shaping, would make it possible to camouflage these details. VPNs could also be combined with MPLS to maximise privacy.

In an ideal case, we should have support from the ISPs to support privacy-aware routing by handling packets according to privacy markings set by the P4 switch. P4 provides full programmability, i.e. traffic handling can be custom-made on standard devices in a vendor-independent way. Suppose that ISP support for privacy is not possible. In that case, VPNs could be established and traffic routed between local (home) and remote (cloud) networks, ideally multiplexing different traffic flows and adding dummy traffic to camouflage traffic patterns.

B. Applicability of the Proposed Framework

Our framework aims to create a data privacy-aware execution zone based on user preferences. A VIM can restrict the execution of an application between two fog units and use other fog units to execute another application. By selectively setting the parameters, it is possible to allow the platform to assume a particular configuration that provides more performance for a specific application.

The proposed framework aims to generate a privacy-aware execution zone and automates the resource termination process. We already mentioned in Section III that we used a simple matching allocation approach, as we assumed that the platform owners already employed better strategies. The resource scheduling process can be more advanced by applying complex or metaheuristics with this framework.

1) Overhead: Now the question arises: *What is the overhead of our framework?* The generation and processing of `INIT` and `STOP` messages, i.e. transmission and processing of small JSON messages, is a simple and lightweight process, including the simple resource selection according to the constraints. Communication with cloud providers via VIMs and actual configuration instantiation and startup of containers/VMs takes most of the time. For example, instantiating a *standard* size VM on a public cloud platform can take two to

three minutes. Therefore, the overhead is low compared to the necessary initialisation of the cloud instance via VIM.

2) Security: Security related to data communication and storage is beyond the scope of this paper. In that case, existing commercial security solutions should be used. For example, VPNs and firewalls can be used to improve communication security. In contrast, encrypted network communication, such as Transport Layer Security (TLS), can be used between the service provider and the user.

C. Learned Lessons

The primary aim of this paper is to validate our framework's functionality and present preliminary results to the community. To our knowledge, existing works are different from ours. It is also worth noting that technical cooperation from ISPs is desirable to transform the proposed prototype into a commercial-grade solution. During the experimentation, we learned about two primary issues:

1) *Performance*: In other related experiments [21], it is found that the end-to-end data packet flow path is not fixed. Network latency, including reliability, may change with changing ISPs. Changing IP protocols from IPv4 to IPv6 can also impact network performance. Even the data packets flow differently inside commercial and non-commercial ISPs.
2) *Scalability*: We have implemented the framework in a P4 software switch to validate the functionality. However, we faced scalability issues, due to the employed software switch. A P4 hardware switch would be the choice for a more extensive setup, but existing P4 hardware switches are challenging to configure.

VII. Conclusion and Long-Term Perspectives

A smart selection of specific platforms (either fog or cloud) to run applications can optimise cost and improve performance. Here, we present a message-based framework to generate an execution zone, according to user data privacy and application preferences, using two JSON-based control messages. The `INIT` message specifies *how to configure the execution environment based on user preferences?*, and *how the data should be handled while in transit?* The `STOP` message releases the held resources while satisfying user preset conditions. Here, we have shown how DSCP-based privacy marking rules could effectively work to create such execution zones. We also prototyped the framework and presented the message structure with eight zone creation scenarios and two resource termination scenarios. Thus, we have validated our approach and made the point that such techniques can be feasible. For a more extensive scope, we need active support from ISPs.

In the future, we will consider multi-cloud ecosystems for dynamically optimising allocations, using orchestration platforms (such as Open Source MANO) to create complex network services. We also plan to make the execution zone 'elastic' to adapt to the dynamic requirements of specific application types, such as distributed deep learning models. We also consider applying a lightweight machine learning model for resource count with resource type prediction to optimise resource modules' allocation.

Disclaimer

The country and company names used in this paper are purely for research purposes and to make our scenarios applicable. No one should infer other meanings (directly or indirectly, explicitly or implicitly) from it.

References

[1] I. Pelle *et al.*, "Towards Latency-Sensitive Cloud Native Applications: A Performance Study on AWS," in *12th International Conference on Cloud Computing*. IEEE, 2019, pp. 272–280.

[2] M. Iorga *et al.*, "The NIST Definition of Fog Computing," National Institute of Standards and Technology, Tech. Rep., 2017.

[3] A. Yousefpour *et al.*, "All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey," *Journal of Systems Architecture*, pp. 289–330, 2019.

[4] F. Staff, "A Look At What ISPs Know About You: Examining the Privacy Practices of Six Major Internet Service Providers," *Federal Trade Commission, Tech. Rep*, 2021.

[5] D. P. Mulligan *et al.*, "Confidential Computing - A Brave New World," in *Int'l Symposium on Secure and Private Execution Environment Design*. IEEE, 2021, pp. 132–138.

[6] R. Guanciale *et al.*, "SoK: Confidential Quartet-Comparison of Platforms for Virtualization-Based Confidential Computing," in *Int'l Symposium on Secure and Private Execution Environment Design*. IEEE, 2022, pp. 109–120.

[7] B. Costa *et al.*, "Orchestration in Fog Computing: A Comprehensive Survey," *ACM Computing Surveys*, vol. 55, pp. 1–34, 2022.

[8] T. L. Duc *et al.*, "Machine Learning Methods for Reliable Resource Provisioning in Edge-Cloud Computing: A Survey," *ACM Computing Surveys*, vol. 52, pp. 1–39, 2019.

[9] X. Masip *et al.*, *Collaborative Mechanism for Hybrid Fog-Cloud Scenarios*. John Wiley & Sons, 2020, pp. 7–60.

[10] H. Lin *et al.*, "A Survey on Computation Offloading Modeling for Edge Computing," *Journal of Network and Computer Applications*, vol. 169, 2020.

[11] R. Landa *et al.*, "Self-Tuning Service Provisioning for Decentralized Cloud Applications," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 197–211, 2016.

[12] G. Castellano *et al.*, "A Service-Defined Approach for Orchestration of Heterogeneous Applications in Cloud/Edge Platforms," *IEEE Transactions on Network and Service Management*, vol. 16, pp. 1404–1418, 2019.

[13] D. Santoro *et al.*, "Foggy: A Platform for Workload Orchestration in a Fog Computing Environment," in *IEEE International Conference on Cloud Computing Technology and Science*. IEEE, 2017, pp. 231–234.

[14] S. Mazumdar and T. Dreibholz, "Secure Embedded Living: Towards A Self-Contained User Data Preserving Framework," *IEEE Communications Magazine*, vol. 60, no. 11, pp. 74–80, 2022.

[15] B. Cheng *et al.*, "FogFlow: Orchestrating IoT Services over Cloud and Edges," *NEC Technical Journal*, vol. 13, pp. 48–53, 2018.

[16] C. Mechalikh *et al.*, "A Fuzzy Decision Tree Based Tasks Orchestration Algorithm for Edge Computing Environments," in *Advanced Information Networking and Applications*. Springer, 2020, pp. 193–203.

[17] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 87–95, 2014.

[18] R. M. Hinden and B. Haberman, "Unique Local IPv6 Unicast Addresses," IETF, Standards Track RFC 4193, 2005.

[19] S. Mazumdar and T. Dreibholz, "Towards a Privacy Preserving Data Flow Control via Packet Header Marking," in *24th Int Conf on High Performance Computing & Communications*. IEEE, 2022, pp. 1509–1516.

[20] P. Biondi and A. Ebalard, "IPv6 Routing Header," in *Proceedings of the CanSecWest Security Conference*, 2007.

[21] T. Dreibholz and S. Mazumdar, "Find Out: How Do Your Data Packets Travel?" in *18th International Conference on Network and Service Management*. IEEE, 2022, pp. 359–363.