

Est.
1841

YORK
ST JOHN
UNIVERSITY

Karthick, Gayathri ORCID logoORCID:
<https://orcid.org/0000-0003-1228-7099> and Mapp, Glenford ORCID
logoORCID: <https://orcid.org/0000-0002-0539-5852> (2024)
Developing a Secure Service Ecosystem to Implement the
Intelligent Edge Environment for Smart Cities. *Future Internet*, 16
(9). p. 317.

Downloaded from: <https://ray.yorks.ac.uk/id/eprint/10633/>

The version presented here may differ from the published version or version of record. If
you intend to cite from the work you are advised to consult the publisher's version:
<https://doi.org/10.3390/fi16090317>

Research at York St John (RaY) is an institutional repository. It supports the principles of
open access by making the research outputs of the University available in digital form.
Copyright of the items stored in RaY reside with the authors and/or other copyright
owners. Users may access full text items free of charge, and may download a copy for
private study or non-commercial research. For further reuse terms, see licence terms
governing individual outputs. [Institutional Repository Policy Statement](#)

RaY

Research at the University of York St John

For more information please contact RaY at ray@yorks.ac.uk



Article

Developing a Secure Service Ecosystem to Implement the Intelligent Edge Environment for Smart Cities †

Gayathri Karthick ^{1,*} and Glenford Mapp ^{2,*}

¹ Department of Data Science and Computer Science, York St John University, Export Building, 1 Clove Cres, London E14 2BA, UK

² Faculty of Science and Technology, Middlesex University, The Burroughs, London NW4 4BT, UK

* Correspondence: g.karthick@yorks.ac.uk (G.K.); g.mapp@mdx.ac.uk (G.M.)

† This paper is an extension of Building an Intelligent Edge Environment to Provide Essential Services for Smart Cities, originally presented at the Workshop on Mobility in Evolving Internet Architecture (MobiArch 2023), Madrid, Spain, 6 October 2023.

Abstract: In the future, smart cities will provide key services including seamless communication, intelligent transport systems, advanced healthcare platforms, urban and infrastructure management, and digital services for local and regional government. Therefore, a new service and networking paradigm, called the Intelligent Edge Environment, has been specified. As a key part of this system, a new secure service ecosystem must be developed to provide the secure real-time movement of services on different High-Performance Edge Cloud Systems. This paper explores these issues by introducing the following mechanisms: a Resource Allocation Algorithm, a Resource Allocation Secure Protocol and finally a Secure Service Protocol. These systems were integrated into the Basic Capability System Library and a multithreaded FUSE client connected to the Service Management Framework. Docker was used as the migration mechanism. A prototype was developed and implemented using a FUSE-Network Memory System in which the Network Memory Server was migrated as users moved around. The result shows that this approach was safe and could be used to develop new applications and services for smart cities.

Keywords: intelligent edge environment; secure service ecosystem; capabilities; FUSE; docker



Citation: Karthick, G.; Mapp, G. Developing a Secure Service Ecosystem to Implement the Intelligent Edge Environment for Smart Cities. *Future Internet* **2024**, *16*, 317. <https://doi.org/10.3390/fi16090317>

Academic Editors: Dimitrios Dechouniotis and Ioannis Dimolitsas

Received: 20 July 2024

Revised: 22 August 2024

Accepted: 24 August 2024

Published: 2 September 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Smart cities need to leverage advanced technologies such as Mobile Edge Computing (MEC), Internet of Things (IoT), AI, Machine Learning (ML), and fast communication networks, for example 5G/6G, to deliver efficient, sustainable, high-quality services, thus improving the overall quality of life for their citizens. These technologies can be combined to produce a new computing and service paradigm, called the Intelligent Edge Environment (IEE), in which services are run and managed from the edge of the network by default. As shown in Figure 1, the IEE is composed of these 7 layers: Heterogeneous Networking Layer, Data Management Layer, the High-Performance Edge Cloud Systems, Service Management Framework, Microservices Layer, Application Framework Layer, and Application Layer. The details of the IEE have been explored in [1].

Intelligent Edge Environment Layers

The layers of the IEE are shown in Figure 2 and are briefly described below:

- Layer 1: Heterogeneous Networking Layer (HNL): This layer provides connections using various wireless technologies such as 4G, 5G, WiFi, and Cellular.
- Layer 2: Data Management Layer (DML): This layer plays a crucial role in handling vast amounts of data generated by edge devices such as Connected and Autonomous Vehicles (CAVs) and uses many structures such as blocks, files, databases, and ML algorithms to manage data.

- Layer 3: High-Performance Edge Cloud Systems (HPECS): This layer supports various cloud architectures and cloud types, including private, public, hybrid, and community clouds (AWS, Hadoop, etc.) using Virtual Machines (VMs) including VMware and Citrix ecosystems as well as video streaming, augmented and virtual reality, and autonomous systems.
- Layer 4: Service Management Framework (SMF): This layer manages services and servers within the system. It offers mobile service support by migrating and replicating services using various migration techniques such as Docker (containerization), KVM (virtualization), and Unikernels (specialized single-purpose virtual machines).
- Layer 5: Microservices Layer (MSL): This layer supports microservices and is responsible for independently deployable services. These services should be fast and small in order to be easily migrated.
- Layer 6: Application Framework Layer (AFL): This layer uses the Microservices Layer below to provide Applications Frameworks (AFs) to build applications for different environments such as Mobile Communications, Vehicular Networking, and IoT Smart Grids.
- Layer 7: Application Layer (AL): This layer allows applications that have been built using the Application Framework Layer to be installed on the system and made available to users. Through this layer, users get applications that use all the resources of the IEE.

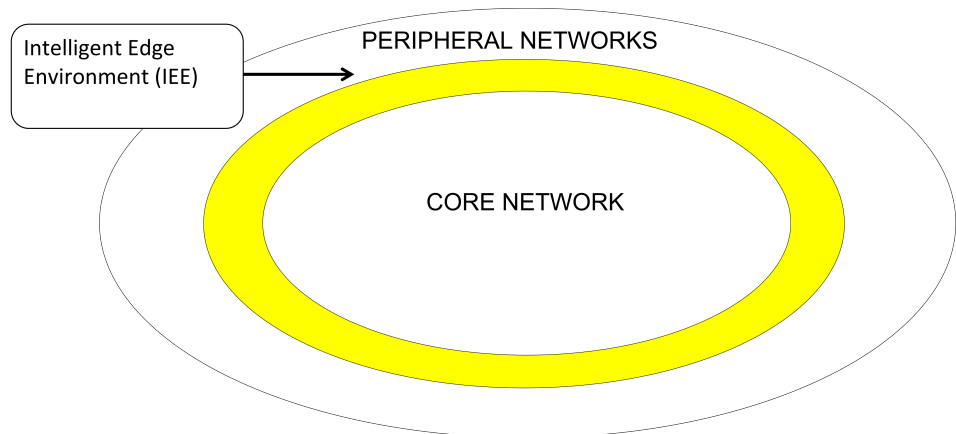


Figure 1. Intelligent Edge Environment.

APPLICATION LAYER
APPLICATION FRAMEWORK LAYER
MICROSERVICES LAYER
SERVICE MANAGEMENT FRAMEWORK
HIGH PERFORMANCE EDGE CLOUD SYSTEMS
DATA MANAGEMENT LAYER
HETEROGENEOUS NETWORKING LAYER

Figure 2. Intelligent Edge Environment layers.

A key part of the IEE is the development of a Secure Service Ecosystem (SSE) in which services can be securely moved between different commercial Cloud Systems. As shown in Figure 2, the SSE, therefore, encompasses the three core layers of the IEE which are the HPECS, SMF, and MSL layers.

This paper explores how to build an SSE and the contributions of the work are detailed below:

- A new simple Resource Allocation Algorithm (RAA) is developed to enable mobile services. This algorithm has to be simple and fast.
- A new secure transfer protocol called the Resource Allocation Secure Protocol (RASP) is detailed. This will involve the use of cryptographic protocol verifiers such as ProVerif to show that the transfer protocol is safe.
- A secure access control system using capabilities to provide the Authentication, Authorization, Accounting (AAA).
- The development of a new Service Management Framework (SMF) which can be used to manage applications and services in the IEE.
- These mechanisms are combined to develop a capability-based Secure Service Protocol (SSP) to provide a complete SSE for the IEE. The SSP is shown to be safe using Proverif.
- The implementation of a prototype SSE system using a Network Memory Server (NMS) to provide backing storage for a FUSE file system in a vehicular network is detailed.

The rest of this paper is organized as follows: Section 2 reviews the Related Work. Section 3 details the Simple Resource Allocation Algorithm, while Section 4 explores the RASP Protocol using ProVerif. In Section 5, a capability system for IEE is investigated, and in Section 6, the SSP Protocol is proposed and tested using ProVerif. Section 7 covers the prototype implementation. Finally, Section 8 concludes the paper.

2. Related Work

2.1. Mobile Edge Computing (MEC)

MEC was originally developed as an offloading mechanism to provide more computing resources at the edge of the network. As shown in [2] there has been a lot of work done in MEC. Two recent projects are EDGELESS [3] and CODECO [4]. EDGELESS is an ongoing EU project that is developing a secure edge-cloud platform that can dynamically adjust itself to ensure a high Quality of Service (QoS) for applications. Although the IEE shares some of these goals, there are significant differences. Unlike EDGELESS, the IEE will manage and run services at the edge of the network by default to support real-time applications with very tight requirements for low latency and high bandwidth; thus, in the IEE there will be no central Cloud platform. The CODECO project is enhancing the container orchestration platform, Kubernetes, with a cognitive edge-cloud management framework that provides support for real-time industrial applications. While CODECO is looking at a software solution to provide real-time support for Kubernetes, the IEE is committed to providing end-to-end low latency using smart networks, caching, and prefetching algorithms as well as low-latency data access and processing via fast hardware, AI and Unikernels.

2.2. Vehicular Networks

The last decade has seen a revolution in transport based on the emergence of Connected and Autonomous Vehicles (CAVs). Several research testbeds have been built [5]. This has spurred more research into Intelligent Transport Systems (ITS) which will result in smaller journey times, less congestion and a reduction in the number of road accidents. Vehicular networks pose serious challenges as they require reliable, low latency, high bandwidth communication as well as the ability to process data in real-time. These challenges can only be met by moving data processing and services to the edge of the network.

The convergence of the automotive industry with cognitive computing, forming the Industrial Cognitive Internet of Vehicles (CIoV), is a rapidly evolving field with significant

implications for the QoS and security in vehicular networks. This convergence is driven by the increasing impact of social media on automotive services and the need for advanced edge computing solutions to reduce latency and enhance reliability. As explored by the author in [6], offloading cognitive computing tasks to the network edge using methods like 88 CQP, which employs Canopy and K-medoids clustering along with a non-dominated sorting 89 genetic algorithm III, can optimize edge server (ES) quantification and placement, thereby improving the QoS by 90%.

2.3. Virtualization and Service Management in Future Internet

Cloud Systems are now used by many companies to provide services to their clients. These systems use virtualisation techniques and enable service migration through Docker, KVM, LXD, and Unikernels mechanisms. Mobile services therefore are a key requirement for future networks. The authors in [7,8], investigated the communication dynamics necessary for seamless connectivity and service migration in vehicular edge environments. The authors in [9] attempted to address the development of a secure commercial advertising scheme for vehicular networks. They proposed a scheme called “Business discovery (BUY)” that presents the concept of the beaconing market using a vehicular network. However, these endeavours also underscore the need for resource management algorithms and secure transport mechanisms to enable mobile services for highly mobile networks.

2.4. Research Gap

Though all these efforts are useful and have led to the development of the IEE platform, new mechanisms and techniques are needed to fully realise this proposed framework. The most pressing of which is the need to develop a secure service ecosystem to allow services to be managed and run from the edge of the Internet by default. This includes a simple and fast resource management algorithm to allow services to migrate using different Edge Clouds, a secure service transfer mechanism, and providing Authentication, Authorisation, and Accounting (AAA) for the new environment. This paper addresses these issues.

3. Simple Resource Algorithm

Resource management [10] is the process of distributing available resources, such as CPU, Memory, Network and Storage, among various tasks in an efficient manner as it helps to deliver high performance in Cloud infrastructure to achieve a high QoS. Cloud systems use virtualization to replicate and share resources. Several algorithms are used to manage this allocation effectively. There are two types of resource algorithms, Static and Dynamic algorithms. However, when services are moved between Clouds, we must use dynamic algorithms which must be simple and fast because it is a highly mobile environment. This algorithm is shown in Figure 3.

The roles in the RAA are specified as follows:

1. Advertising Cloud (CA, CB, CC... etc.): Each Cloud Service Provider (CSP) has a finite set of resources available for services and servers. A Cloud System broadcasts its available resources (CPU, Memory, Network, and Storage) to the Receiving Servers.
2. Receiving Servers (SA, SB, SC... etc.): These servers can choose to migrate their services to the Advertising Clouds based on their hosting requirements.
3. Resource Allocation Server (RAS): A trusted party responsible for verifying the resources of Advertising Clouds and the hosting requirements of Receiving Servers. Table 1 shows the information held by the RAS.

The first stage of the RASP protocol uses the Resource Allocation Algorithm (RAA) for cloud advertisements. Each Cloud System actively advertises its free resources in each round and uses mathematical formulations to determine if a service can be securely migrated to an Advertising Cloud (AC). The formulation below demonstrates how Cloud CB advertises its resources and Receiving Server SA obtains the resources it needs to run on the AC.

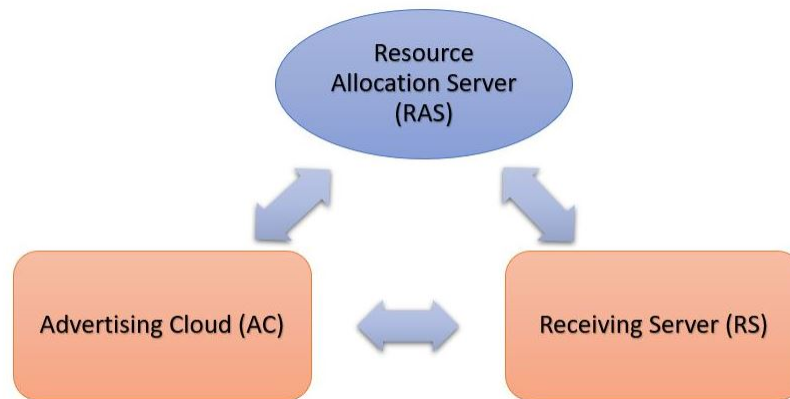


Figure 3. Simple resource allocation design.

Table 1. RAS Table.

Resource Allocation Server	Adverting Cloud (CA)	Adverting Cloud (CB)	Adverting Cloud (N)
Receiving Server (SA)	CPU, Memory, Network, and Storage	CPU, Memory, Network and Storage	CPU, Memory, Network, and Storage
Receiving Server (SB)	CPU, Memory, Network, and Storage	CPU, Memory, Network, and Storage	CPU, Memory, Network, and Storage
Receiving Server (N)	CPU, Memory, Network, and Storage	CPU, Memory, Network, and Storage	CPU, Memory, Network, and Storage

3.1. Advertising Cloud Formulation

The variables indicate the number of CPUs, Memory, Storage, and Network resources respectively. These values help in determining the resource availability and allocation for secure service migration. At each round, each provider will use the mathematical formulation below: Step 1 of the RASP protocol and general notations is as follows:

- Stage 1 1.** CB → SA: Advc (CB, ResCB) [CB → SA: Advertising Cloud (CB) sends an advertisement (Advc) to the Receiving Server (SA), detailing its free resources (ResCB)].
 - Maximum Resource of Cloud (MAX_Cm) = [CPU (Cmc), Memory (Cmm), Storage (Cms), Network (Cmn)],
 - Allocated Resource of Cloud ($ALLOC_Ca$) = [CPU (Cac), Memory (Cam), Storage (Cas), Network (Can)],
 - Free Resource of Cloud ($FREE_Cf$) = [CPU (Cfc), Memory (Cfm), Storage (Cfs), Network (Cfn)]

Cloud CB Formulation Algorithm 1 in detail,

Algorithm 1 Cloud CB Formulation

- Resources(r) ∈ {Cpu(c), Network(n), Memory(m), and Storage(s)}
- MAX_Cm (Cmr) = {Cmc, Cmm, Cms, Cmn }
- $ALLOC_Ca$ (Car) = { Cac, Cam, Cas, Can } &
- $FREE_Cf$ (Cfr) = {Cfc, Cfm, Cfs, Cfn}
- $Cfr \leftarrow Cfr \cup Resources(r)$
- If $Cfr \leftarrow (Cmr - Car) > 0$ then
- Send Advc of Cfr and wait for response;
- else $Cfr \leq \phi$ then
- wait for $Cfr > 0$;
- End if

- **Step 1:** The resources in consideration are categorized as CPU (c), Network (n), Memory (m), and Storage (s).
- **Step 2–4:** These steps define the maximum capacity for each type of resource (C_{mr}), current allocation of resources (C_{ar}), and the available (free) resources (C_{fr}) in the AC.
- **Step 5–10:** Condition Check and Advertisement: The condition checks if there are any free resources (C_{fr}). If there are free resources, the AC sends an advertisement of these resources and waits for a response. If there are no free resources, the system waits until resources become available, i.e., when (C_{fr}) is greater than 0.

3.2. Receiving Servers Formulation

Receiving Servers (RS) receive an advertisement from the Advertising Cloud (AC) and its free resources. A server that needs to use a cloud has variables that detail its requirements. If the free resources of AC, C_{fc} , C_{fm} , C_{fs} , C_{fn} are larger than Receiving Server requirements, it can forward the request to the Resource Allocation Server (RAS) also known as the Registry (R). The required resources of the server are given by (Req_Res of SA (S_{Areqr})). If we assume the requested resources a less than the free resources of the AC and all requested resources are valid, then and only then, it sends the request to RAS. Step 2 of the RASP protocol is as follows:

- **Stage 2 2.** SA \rightarrow R : (SA, CB, ResCB)pkR [SA \rightarrow R: Receiving or Requested Servers (SA) send a message to the Registry to verify the identity and free resources (Res freeCB) of the Advertising Cloud, CB].
 - Requested Resource of Server (Req_Res of SA (S_{Areqr})) = {Src, Srm, Sms, Srn} CPU (Src), Memory (Srm), Storage (Sms), Network (Srn)]

Server SA Formulation Algorithm 2 in detail,

Algorithm 2 Server SA Formulation

- 1: Resources(r) \in {Cpu(c), Network(n), Memory(m), and Storage(s)}
 - 2: Req_Res (S_{Areqr}) = {Src, Srm, Sms, Srn}
 - 3: $S_{Areqr} \leftarrow S_{Areqr} \cup Req_Resources$
 - 4: If $S_{Areqr} > 0$ then
 - 5: Search for Resource Migration;
 - 6: If $S_{Areqr} \leftarrow (Cf - S_{Areqr}) > 0$ then
 - 7: Accept Cf and send a message to RAS;
 - 8: Else ignore the request;
 - 9: End if
-

- **Step 1:** The resources CPU, Networks, Memory, Storage are members of resources(r).
- **Step 2–3:** Server SA's Requested resources are declared as a set of elements and is given by S_{Areqr} resources.
- **Step 4–5:** If the Requested resources are greater than 0, the server can look to migrate the service to the Advertising Cloud.
- **Step 6–9 Condition check before resource Migration:** It checks if the AC has sufficient free resources to meet the Receiving Server's request. If sufficient resources are available, then the request is granted, and the details are sent to the RAS for verification and processing, else Server SA ignores the advertisement.

3.3. Resource Allocation Server

The Resource Allocation Server (RAS) plays a crucial role in the resource allocation process. It is a trusted Party responsible for confirming the registration of Cloud Providers and certifying the capacity of their resources. The RAS ensures that only valid and adequately resourced Clouds participate in service migrations. Function of RAS in the Algorithm are:

- **Registration and Certification:** The RAS confirms the registration of Cloud Providers and certifies the capacity of resources available with the registered Cloud Providers.

- **Request Verification:** When a request for resources is received from a Receiving Server (SA), the RAS verifies the resources based on specific criteria before approving the migration.

Stage 2 of the RASP protocol therefore provides verification and certification. Once it receives the request from SA requested resources ($SAreqr$), the RAS verifies the resources based on the Algorithm below:

- 3. $R \rightarrow SA: \text{sign}((CB, pkC, ResCB), pkS)$ [$R \rightarrow SA$: The Registry approves the advertisement ($AdvC$) and sends a message to the Receiving Servers (SA), detailing the free resources ($ResCB$) with its signature].

Server RAS Formulation Algorithm 3 in detail:

Algorithm 3 RAS_CB

- 1: If $Cf = \text{Valid Cloud}$ and $RAS \leftarrow (Cf > SAreqr) \neq 0$ then
 - 2: Accept and send an authenticated message;
 - 3: Else
 - 4: Reject the request;
 - 5: End if
-

- **Step 1–2 Verify Validity:** The RAS server verifies whether the CB is a valid cloud and if its Cloud resources are greater than the requested resources. If either condition is false, then it rejects the request and updates Server SA that CB is an invalid cloud.
- **Step 2–5 Pass Conditions:** If the above conditions pass, then it sends an authenticated message to SA. This message confirms the approval of the resource migration.

Server RAS Algorithm 4 Formulation in detail,

Algorithm 4 RAS_SA

- 1: If $SAreqr = \text{valid server}$ and $RAS \leftarrow (Cf > SAreqr) \neq 0$ then
 - 2: Accept and send an authenticated message;
 - 3: Else
 - 4: Reject the request;
 - 5: End if
-

- **Step 1–2:** RAS server verifies whether it is a valid server and if Cloud resources are greater than the requested resources which is not equal 0. If either conditions are false, then it rejects the request and updates SA that its an invalid Cloud.
- **Step 2–5:** If the above conditions pass, then it sends an authenticated message to SA.

The RAS is essential for ensuring the integrity and efficiency of resource allocation in Cloud environments. It validates Cloud Providers, certifies their resources, and manages requests from Receiving Servers, ensuring only valid and adequately resourced Clouds participate in service migrations. This multi-step verification process helps maintain a secure and reliable resource allocation framework.

4. RASP Protocol—An Overview

This protocol has three roles: Server SA on Cloud CA, Cloud CB, and Registry. The Resource Allocation Security protocol (RASP) is broken into four stages. In the first stage, it uses the Resource Allocation Algorithm to manage cloud advertisements [11]. In addition, the proposed protocol was tested by ProVerif. The protocol is shown in Figure 4 as follows: Stage 1 corresponds to step 1 of the protocol; Stage 2 corresponds to steps 2–7; Stage 3 corresponds to steps 8–11, and finally, Stage 4 corresponds to step 12. The RASP protocol is followed in exactly the same way as outlined below and shows the steps for Cloud-to-Cloud migration of services.

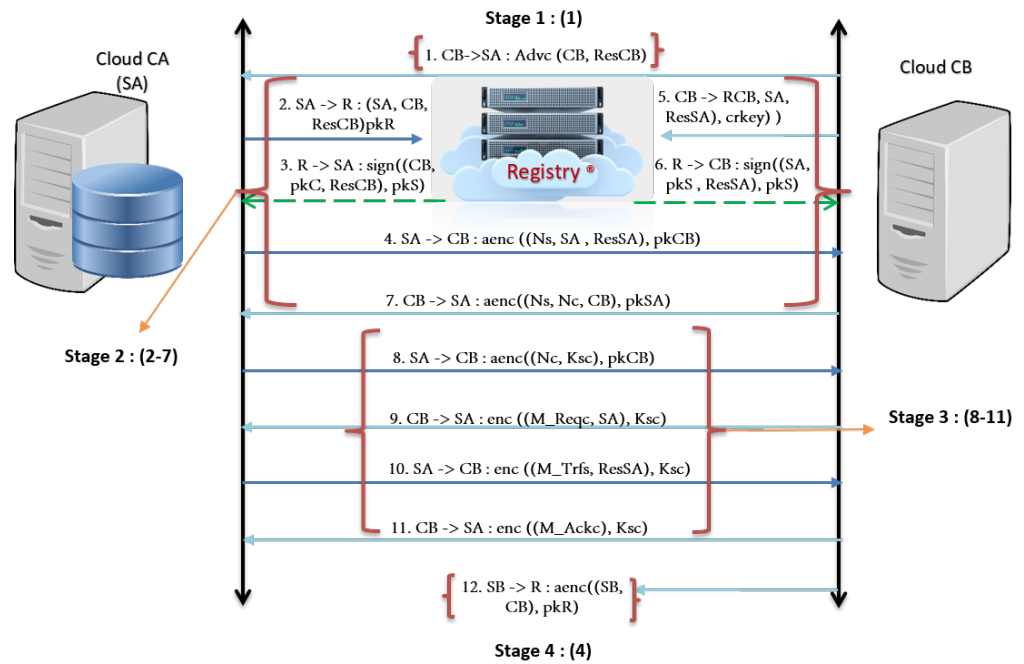


Figure 4. Migration between SA to CB.

4.1. General Notations

Table 2 shows the parameters used in the RASP protocol.

Table 2. RASP Table.

Notation	Explanation
CA, CB, SA and SB	Cloud and Server Identities
pkC/skC, pkS/skS, pkR/skR	public and private key pairs of Cloud CB, Server SA & Registry
M_Reqc, M_Trfs, M_Ackc	Request for migration, transfers and acknowledgement
sign	Signature/signed by the Registry
aenc	Asymmetric Encryption
enc	Symmetric Encryption
Ksc	Symmetric session key
Ns, Nc	Nonce of SA and CB
ResSA, ResCB	Requested Resources of SA & CB

- Stage 1: Advertisement :**
 Server SA receives advertisements from Cloud CB advertising its resources.
 1. CB → SA: Advc (CB, ResCB)
- Stage 2: Authentication of SA and CB as well as migration request and response:**
 Server SA first requests the RAS/Registry to authenticate Cloud CB and its resources. In Step 3, the Registry (R) authenticates Cloud CB. In Step 4, Server SA sends a migration request to Cloud CB. In Step 5, Cloud CB sends a request to the Registry to verify that Server SA’s request for resources is valid. In Step 6, the Registry replies to Cloud CB. In Step 7, Cloud CB sends the migration response back to Server SA.
 2. SA → R: (SA, CB, ResCB)pkR
 3. R → SA: sign((CB, pkC, ResCB), pkS)
 4. SA → CB: aenc((Ns, SA, ResSA), pkC)
 5. CB → R: (CB, SA, ResSA)pkR
 6. R → CB: sign((SA, pkS, ResSA), pkC)
 7. CB → SA: aenc((Ns, Nc, CB), pkS)

3. Stage 3: Migration transfer:

In Step 8, Server SA generates the session key (Ksc) to start the migration. In Step 9, Cloud CB sends the migration initialization request. In Step 10, Server SA performs the migration transfer. In Step 11, Cloud CB sends an acknowledgment to Server SA.

8. SA → CB: $\text{aenc}(\text{Nc}, \text{Ksc}), \text{pkC}$

9. CB → SA: $\text{enc}((\text{M_Reqc}, \text{SA}), \text{Ksc})$

10. SA → CB: $\text{enc}((\text{M_Trfs}, \text{ResSA}), \text{Ksc})$

11. CB → SA: $\text{enc}((\text{M_Ackc}), \text{Ksc})$

4. Stage 4: Update of New service location to the Registry:

In Step 12, the service on Cloud CB (SB) updates the Registry on its new location. The new service SB is now running on Cloud CB and informs the Registry that it has been successfully migrated.

12. SB → R: $\text{aenc}((\text{SB}, \text{CB}), \text{pkR})$

4.2. ProVerif an Overview

ProVerif [12] is a formal verification and highly automated tool used for analyzing the security properties of cryptographic protocols. It employs applied calculus to model concurrent systems and their interactions. The tool comprehends the protocol's abstraction, including how messages are exchanged between parties and the use of cryptographic primitives such as encryption, decryption, and signatures. ProVerif verifies the service migration mechanism's secrecy, authentication, and key exchange. Another strength of this tool is that it performs symbolic analysis, aiding in understanding attack scenarios. ProVerif provides attack traces [13] if it finds any violations of a security property.

The output of the Query attacker() query function results as "True", indicating that the security property cannot be accessed by the attacker. This validates the attacker's lack of knowledge and ensures the secrecy of data authentication between parties or multiple roles for securely transferring data and authenticated across an unlimited number of sessions using unbounded data. If the outcome of the attacker() function is "False", the security property value is accessible to the attacker. Finally, the result displays any attack trace and indicates whether the attacker() query returns TRUE or FALSE.

ProVerif Results

The results shown in Table 3 indicate that RASP can preserve the secrecy, authentication and key exchange of the service migration mechanism.

- **Nonces are secured and not derived by the attacker:** specific nonces (SNs, SNc, CNs, CNc) used in the protocol are not derivable by an attacker, ensuring their confidentiality. Each line indicates that the respective nonce cannot be compromised (is true).
- **Session key:** Results confirm that the session key (Ksc) and associated bitstrings (SNk, CNk) are secure from the attacker. The line "RESULT not attacker_bitstring (SNk []) is true" with Ksc also specifies the conditions under which this key is secure.
- **Private keys:** Results confirm that the session key (Ksc) and associated bitstrings (SNk, CNk) are secure from the attacker. The line "RESULT not attacker_bitstring (CNk []) is true" with Ksc also specifies the conditions under which this key is secure.
- **Authentication:** It validates the mutual authentication between SA and CB. The use of inj-event indicates that the protocol verifies injective correspondence, meaning each event endSparam (or endCparam) has a matching beginSparam (or beginCparam), ensuring that both parties authenticate each other correctly.

By using a symmetric session key (Ksc), the requested service is transferred to the new location CB. This section has explored the development of a new resource allocation and secure service migration framework that encompasses the computing resources to migrate services in cloud environments.

Table 3. Query attacker Results

Security Properties	Server SA Event	Cloud CB Event
Session key	True	True
Private keys	True	True
Nonces	True	True
Event begin	True	True
Requested resources	True	-
Advc Resources	-	False (Not encrypted)

5. Capabilities

Capabilities are critical for system security, providing a robust mechanism for access control through unforgeable tokens. It plays a crucial role in identifying objects and their properties within a system. It is essential to manage and protect these capabilities diligently to prevent unauthorized generation or alteration. Proper management is vital to maintaining system integrity and security, ensuring that only authorized users can access specific objects and perform actions according to their assigned capabilities. This careful oversight helps safeguard the system against potential security breaches and misuse. The data structure contains two pieces of data and functionality.

- **Data:** Unique Object Identification and Access Rights.
- **Functionality:** Capabilities can provide Role-Based Access Control (RBAC) access for users. Some capabilities are not directly assigned to users; instead, they are assigned to roles, and roles are then assigned to users. These are known as role-based capabilities. This structure supports RBAC, ensuring that users have access based on their roles within an organization.
- **IPv6 Address Space and Capability ID System:** Utilizes a modified Location/ID split based on the work of [14]. This enables the creation of a capability ID-based system for people, applications, and cloud infrastructure.

5.1. Capabilities in the IEE

Intelligent Edge Environment (IEE) capabilities are crucial for secure and efficient access control. By utilizing role-based capabilities and advanced addressing systems like IPv6, the IEE can ensure robust access management and resource allocation while maintaining high levels of security and integrity. The IEE is a dynamic environment where security challenges are significantly higher than those in traditional computing environments. Traditional AAA (Authentication, Authorization, and Accounting) mechanisms, such as using RADIUS servers, are no longer effective solutions in this context. Given the scalability requirements of large systems, it was decided that AAA should be based on the subject or user rather than the object. This approach is more scalable and suitable for modern, complex environments. Consequently, capabilities are used to provide AAA for the IEE, ensuring that security and access control mechanisms are effective and adaptable to the unique demands of edge computing [15].

Capabilities Structure

In the Intelligent Edge Environment (IEE), every object and its properties are identified using capabilities. These capabilities must be carefully managed and protected to prevent unauthorized creation or modification, and they should be easily revocable. The format of the capability-based system is shown in Figure 5, and its structure is explained below:

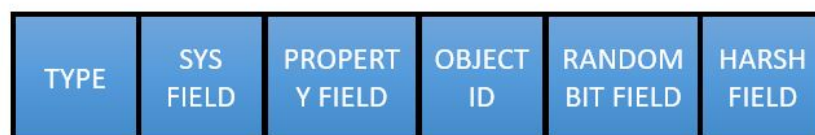


Figure 5. Capability structure.

- **Type Field (8 bits):** This field specifies the type of object capability being used, such as users, digital assets, facilities, etc.
- **SYS Field (4 bits):** This field helps manage capabilities. The four bits within the SYS field are explained below.
- **Property Field (12 bits):** This field defines the properties of the object associated with the capability. It relates to the properties or functions of the object that the capability refers to.
- **Object ID (72 bits):** This field uniquely identifies the object in the system. It includes a EUI-64 identification field to identify the object and an 8-bit netadmin field to manage the object on a network.
- **Random Bit Field (16 bits):** This field provides unforgeability and helps uniquely identify the object. This field is generated after the type field, SYS field, property field, and Object ID field are created. When proxy certificates are created, a new random field is generated. This field also allows for easy revocation of capabilities by simply changing the random field and recomputing the hash field, hence revoking previous versions.
- **Hash Field (16 bits):** This field detects the tampering of capabilities. When a capability is created, the type field, SYS field, property field, and Object ID field are first generated, followed by the random bit field. These fields are then used to generate a SHA-1 hash, which is placed in the Hash Field of the capability.

As shown in Figure 6, the SYS field consists of the following bits:

- **Private or P bit:** Restricts the list of people holding the capability. With a public capability, only the capability for the object must be presented, allowing anyone to hold it without needing the identification of the subject, the person holding the capability. With a private capability, both the object's capability and the subject's capability must be presented to ensure the subject has the right to invoke the object.
- **System or S bit:** Indicates whether the object involved was created by the system or by an application or user. A system capability cannot be modified or deleted by users or applications.
- **Master or M bit:** Indicates that the capability was created by a Certificate Authority (CA). The master capability is usually created when the object is created. If this bit is not set, it means this is a proxy capability. Proxy capabilities are derived from master capabilities and cannot be derived from other proxy capabilities.
- **Change or C bit:** Indicates whether this capability can be changed. If this bit is set, proxy capabilities can be derived from the master capability. If this bit is not set, the capability cannot be modified, and proxy capabilities cannot be generated.

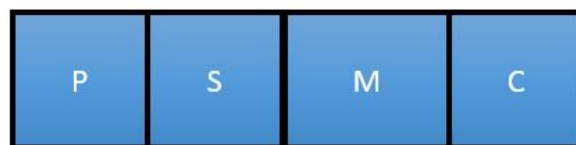


Figure 6. Capability structure—SYS FIELD.

By adhering to these principles, Intelligent Edge Environments can maintain a high level of security and efficiency, ensuring that resources are managed effectively and that unauthorized access is prevented.

6. The Secure Service Protocol (SSP)

The advancement of highly mobile technologies such as Vehicular Adhoc Networks (VANETs), MEC, AI, ML, and IoT systems requires multiple interfaces to seamlessly connect and provide optimal performance for mobile users. As discussed in the Introduction, MEC facilitates moving services closer, minimizing the need for data processing on remote servers, reducing delays, and increasing network bandwidth. However, in order to allow

services to run from the edge of the network, it is essential to enable service migration to support these networks, allowing services to migrate as mobile users move around. Figure 7 is a visual representation of the change needed. In the old environment, the service provider had to manage the scaling and migration of services. In the new environment, these functions are now managed by the Service Management Framework (SMF).

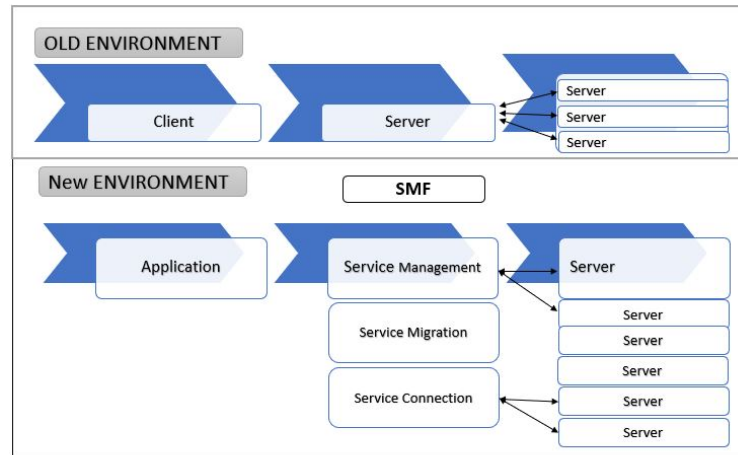


Figure 7. Effect of introducing SMF in the client–server environment.

A new SMF for mobile services has been developed, as shown in Figure 8 and detailed in [16].

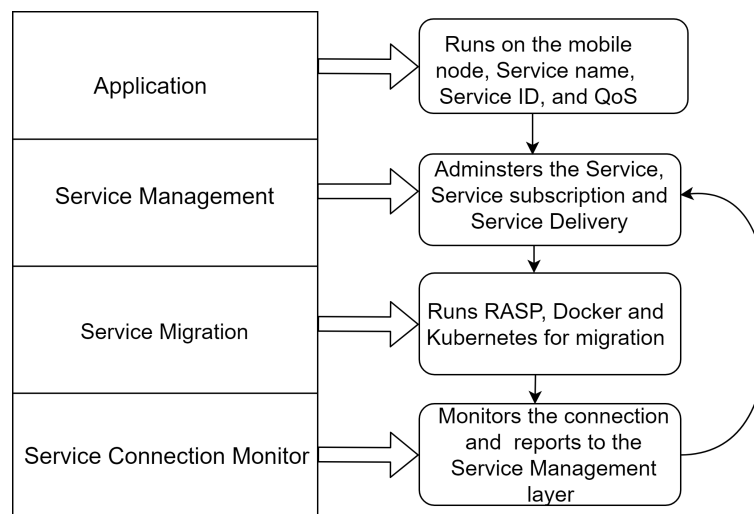


Figure 8. Service Migration Prototype (SMP).

6.1. SMP Layers in Detail

- The Application Layer (AL) is the first layer of the SMF. It runs on the mobile node and calls the service through the Service Management Layer (SManL). AL provides the service name, Service ID, and the required QoS of the service it requires such as delay time, latency, bandwidth, reliability, and security. When a service first registers with the SMF, it is given a unique ID for identification and a service name which indicates the type of service being provided, and the resources such as CPU, Memory, or Storage needed to run the service.
- The Service Management Layer (SManL) is responsible for administering the mobile service, managing service subscriptions, and overseeing service delivery for the layers below it. Service subscription includes managing Service-Level Agreements (SLAs) and billing. Service delivery entails determining how and when services should move

from one location to another. When this layer determines that a service should be migrated, it forwards this information to the Service Migration Layer (SML).

- The Service Migration Layer (SML) manages migration requests from SManL and employs RASP to securely execute the migration process. RASP, as described in [17], utilizes standard migration mechanisms like Docker, KVM, LXD, and Unikernels for the actual migration. Upon completion, SML informs SManL.
- Service Connection Layer (SCL): This layer monitors the connections between the mobile node and the server. It reports to the SManL when the mobile node is no longer available due to handover to another network.

6.2. SSP Protocol—An Overview

Capabilities can be flexibly used to provide AAA in many environments. Furthermore, by combining capabilities, SMF, and RASP techniques, it is possible to design a Secure Service Protocol (SSP) that can protect any service. The SSP protocol has been developed to ensure that the proposed service protocol is secure and can be applied to any service. It is broken down into five stages to clarify the necessary operations in secure service migration. The interaction between an application, the Service Management Framework (SMF), and services supported by the SMF are shown in Table 4 and Figure 9. The SSP protocol is defined in the same way as the RASP protocol outlined above.

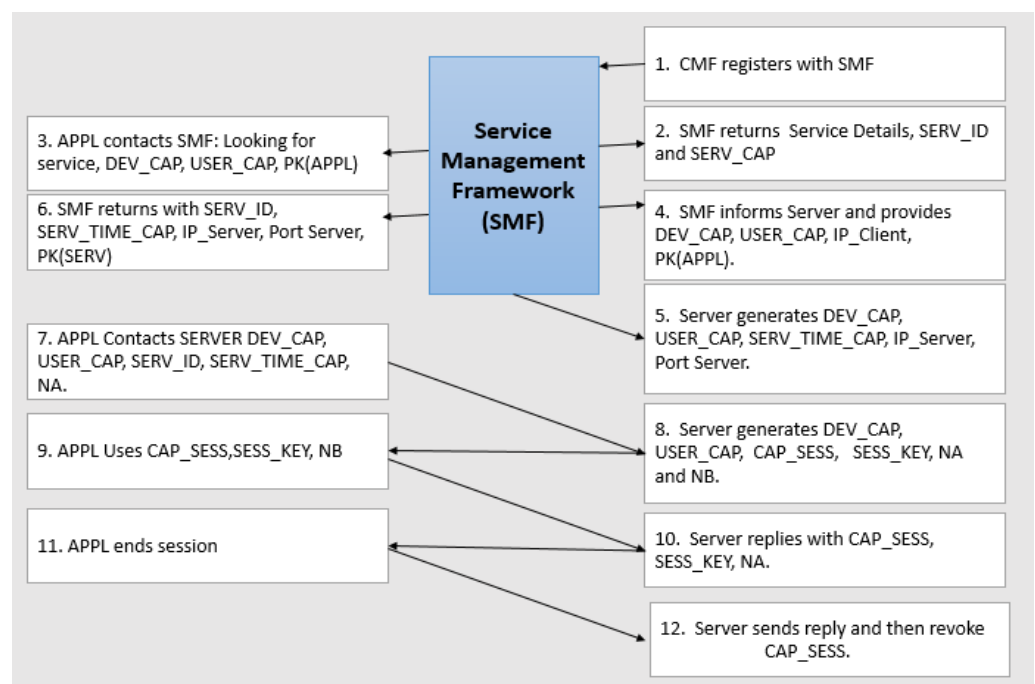


Figure 9. Secure service protocol.

Table 4. General Notations.

Notations	Explanation
CMS	Cloud Management System
SMF	Service Management Framework
APPL	Applications
Chosen Server	The SMF chooses a server of that service
PK(SMF)/PK(APPL)/PK(SERV)	Public key of SMF/APPL/Server
SK(SMF)/SK(APPL)/SK(SERV)/	Private key of SMF/APPL/Server
SESS KEY	Session key

Table 4. Cont.

Notations	Explanation
NA and NB	Nonce of NA and NB
Register Service	CMS contacts to Register the Service
Service Details	Servers, keys, and running services.
Service Registered	The SMF registers the Service
SERV ID	Service ID
SERV CAP	Service Capability
DEV CAP	The Capability for the Device
Request service	Request to an appropriate server
USER CAP	User capability
CAP SESS	Session capability
Service Name	Name of the service
Service Version	Service Version
IP Client	IP address of Client
SERV TIME CAP	proxy capability
IP Server	IP address of Server
Port Server	TCP or UDP Port of Service
Usage request	SMF contacts the server
Usage request accepted	The Server accepts the request
Request service accepted	SMF returns all the server details
Session Start Request	Time capability sends start request
Service request	The application requests service
Service request done	The application gets service
Request Details	Details of APPL request
Result Details	Details of Chosen Server result
Session End Request	The application terminates the session
Session End Req accepted	Server terminates session

- Stage 1: Registration of the Service with the SMF:** In Stage 1, the Cloud Management System (CMS) registers the service with the SMF. This involves providing a list of servers and their public keys that will be running the service. After this, in step 2, the SMF registers the service and returns the Service ID (SERV ID) and the Service Capability (SERV CAP). Following this, the CMS initiates the servers to implement the Service on different machines and passes the Service ID and the Service Capability.
 - CMS → SMF: (Register Service (Service Details), (PK(SMF)))
 - SMF → CMS: (Service Registered (Service Details, SERV ID, SERV CAP), (PK(CMS)))
- Stage 2: The Application interacts with the SMF to get a server that implements the Service:** In Stage 2, the Application interacts with the SMF to obtain a server that provides the service. To utilize a service, an application must request the SMF to locate a suitable server. In step 3, the Application provides the Device Capability (DEV CAP), User Capability (USER CAP), Service Name, Service Version, and the Application's public key running on the machine. In step 4, the SMF selects a server for that Service and contacts it using its public key, passing the Client's DEV CAP, USER CAP, and IP address. In the subsequent step, the Chosen Server accepts the request. It generates a timed capability from the service capability, which expires

in 60 s if the Application does not connect to the server. This SERV TIME CAP is a private proxy capability derived from the Service Capability and cannot be altered; only the USER CAP and DEV CAP can utilize the capability. The Chosen Server also provides its IP address and the TCP port of the Service using the public key of the SMF. In the final step of Stage 2, the SMF returns to the Application with DEV CAP, USER CAP, timed capability, the Server's IP address, Service Port number, and public keys of the Chosen Server, enabling the Application to connect to the server.

3. APPL → SMF: (Request service (Service Name, Service Version, DEV CAP, USER CAP, PK(APPL)), (PK(SMF)))
 4. SMF Chosen → Server: (Usage request (DEV CAP, USER CAP, IP Client), (PK(SERV)))
 5. Chosen Server → SMF: SMF (Usage request accepted (DEV CAP, USER CAP, SERV TIME CAP, IP Server, Port Server), (PK(SMF)))
 6. SMF → APPL: (Request service accepted (DEV CAP, USER CAP, SERV TIME CAP, IP Server, Port Server, PK(SERV)), (PK(APPL)))
- **Stage 3: The Application sets up a secure session with the Server:** During Stage 3, the Application initiates a secure session with the Server by sending a session start request. This request includes the DEV CAP (device capability), USER CAP (user capability), SERV ID (server ID), a unique random number called Nonce of A (NA), the Server's public key, and the SERV TIME CAP (server time capability). Nonce (NA) ensures that the session is unique on the Application side. In response, the Server sends back a session capability (CAP SESS), DEV CAP, USER CAP, NA, NB (another unique random number for the Server side), and a session key (SESS KEY). The SESS KEY is a symmetric key used in a single communication session.
 7. APPL → Chosen Server: (Session Start Request (DEV CAP, USER CAP, SERV ID, SERV TIME CAP, NA)), (PK(SERV)).
 8. Chosen Server → APPL: (Session Start Request Accepted DEV CAP, USER CAP, CAP SESS, SESS KEY, NA, NB), (PK(APPL)).
 - **Stage 4: The Application gets service by using the CAP SESS key and encrypts using the SESS KEY:** In Stage 4 of the process, the application uses the CAP SESS key to access a service and encrypts data using the SESS KEY. The application sends a service request to the chosen server with the CAP SESS, Nonce (NB), and uses the SESS KEY. The server responds by sending CAP SESS, NA, and detailed results using the SESS KEY. NA ensures that the session is unique on the application side.
 9. APPL → Chosen Server: (Service Request (CAP SESS, NB, Request Details) (SESS KEY))
 10. Chosen Server → APPL: (Service Request Done (CAP SESS, NA, Result Details) (SESS KEY))
 - **Stage 5: The Application is finished and terminates the session:** In Stage 5, the application terminates the session by sending an end request to the Chosen Server with CAP SESS, nonce (NB), and the session end indication using the session key. The server then terminates the session and revokes the CAP SESS capability to prevent replay attacks.
 11. APPL → Chosen Server: (Session End Request Accepted (CAP SESS, Nonce (A), Session End)(SESS KEY)).

6.3. ProVerif Results

The SSP protocol was designed using the ProVerif tool, and the results show that the system is "safe". In Stage 4, the application gets service by using the capability session key and encrypts using the Session key to the Chosen server. The query attacker of function to "Session key" (Ksappl) is TRUE.

The private keys of Cloud Management System (CMS), Service Management Framework (SMF), Application (Appl), and Chosen Server(S) results as True. In Stage 3 and

Stage 4, nonces are used between the chosen Server and the Application to prevent replay attacks. The output of nonce is “True” from Server sessions. The Cloud Management System (CMS), which supports a number of services, contacts the SMF to register the service. We applied the “event begin and end function” to verify whether the sessions started before the SMF started or not, and the result is “True”. In the same way, the event begin and end functions are used to verify the server sessions.

Table 5 shows that the SSP protocol is working fine. The results from ProVerif are displayed in Figure A1 in the Appendix A. Hence, this is a more secure mechanism for capability-based services.

Table 5. QUERY ATTACKER Results using ProVerif

Security Properties	CMF	SMF	SERVER	APPL
Private Keys	True	True	True	True
Session Key	-	-	True	True
Nonces	-	-	True	-
Event Begin()	True	-	-	-
Event end()	True	-	-	-
inj-event begin()	-	-	True	-
inj-event-end()	-	-	True	-

6.4. Queries for Private Keys

Queries for Private keys of Application, Cloud Management System, Service Management Framework and Server : *query attacker(sksmf)*; *query attacker(skcms)*; *query attacker(skappl)*; *query attacker(skS)*;

- RESULT not attacker_skey(sksmf[]) is true.
- RESULT not attacker_skey(skcms[]) is true.
- RESULT not attacker_skey(skappl[]) is true.
- RESULT not attacker_skey(skS[]) is true.

6.4.1. Queries for nonces

Queries for nonce between Server and application: *query attacker(SERVERNA)*; *attacker(SERVERNB)*; *attacker(APPLNB)*; *attacker(APPLNA)*;

- RESULT not attacker_bitstring_(SERVERNA[]) is true.
- RESULT not attacker_bitstring_(SERVERNB[]) is true.

6.4.2. Queries for Symmetric key

Queries for Symmetric key : *query attacker(Ksappl)*; *query attacker(SERVERk)*

- RESULT not attacker_key (Ksappl[m7 = v_10131,m4 = v_10132,!1 = v_10133]) is true.
- RESULT not attacker_bitstring (SERVERk []) is true.

6.4.3. Queries for authentication of Server event and CMS event

Queries for Symmetric key : *query attacker(Ksappl)*; *query attacker(SERVERk)*

- RESULT inj-event(endSERVERparam(x_24696)) → inj-event(beginSERVERparam(x_24696)) is true.
- RESULT inj-event(endCMSparam(x_38772)) → inj-event(beginCMSparam(x_38772)) is true.
- RESULT event(endCMSparam(x_42655)) → event(beginCMSparam(x_42655)) is true.

7. Prototype Implementation

Using new mechanisms such as Capabilities, and a new Service Management Framework, this research has proposed a new Secure Service Protocol, which can be used to protect any service. The protocol has been verified using ProVerif. In this section, we explore the building of a prototype system.

7.1. Basic Capability System Library (BCSL)

We developed our prototype system using the Basic Capability System Library (BCSL) as our foundation. This library, depicted in Figure 10, is part of the Base Layer of the Implementation Framework. It supports the creation of users, devices, and services and allows servers to be added to services. Each component has its capability, which is used to access other objects, such as files. Services are created and registered with the Service Management Layer (SManL). The BCSL created by the ALERT Team of Middlesex University, London, is a primary system that makes users, devices, services, and servers. These types are identified as CAP USER, CAP DEVICE, and CAP SERVICE as their Capability types, respectively.

The middle layer is responsible for enhancement, tuning, mapping layers, and updates while the top layer focuses on system implementation. In this layer, our primary goal is to improve the Service Management Framework (SMF), so that it can securely replicate and migrate services.

Implementation Framework

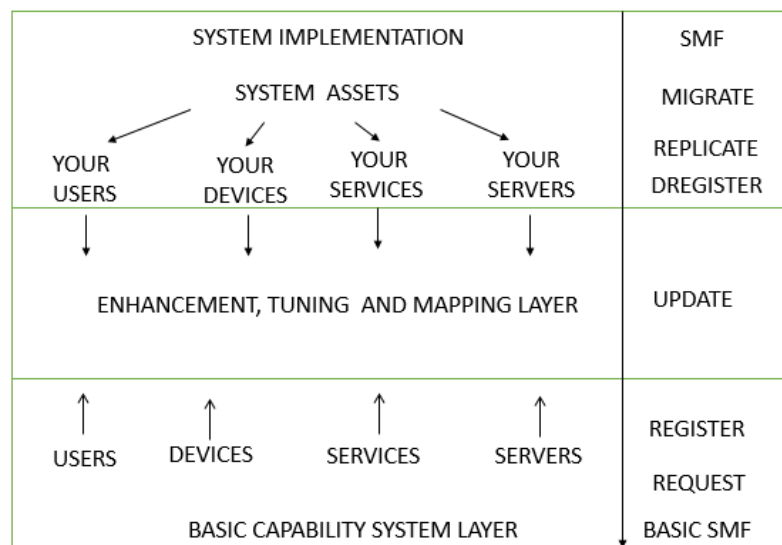


Figure 10. Basic Capability System Library (BCSL).

7.2. Make Fuxfs Multithreaded

The FUSE and NMS servers store and manage the system’s data. The NMS provides networked block storage. When a client moves to another network, the NMS is migrated to the new network to improve performance. In this scenario, the client is referred to as “fuxfs”, and the server is called “fuxfs server”, and they communicate using normal communication protocols.

7.3. The NMS Server

The NMS server is a simple program that provides networked block storage; so should be easy to migrate between networks. This NMS server is working with the FUSE system. Hence this executable was called the Fuxfs_server.

7.4. The FUSE Client

This is the client side of the FUSE code and runs on the mobile node. The executable was called “Fuxfs” and interacts with the Fuxfs_server to manage and store data.

7.5. NMS Part of FUSE as a Service

This framework can be used to implement microservices in vehicular environments. Microservices are faster to migrate and hence can be used to maintain QoS in these networks. As our first use case, we consider a FUSE file system that is a user space file system commonly employed in the Linux environment. The setup, which is shown in Figure 11, provides a storage platform for mobile users as the NMS provides block network storage to FUSE clients which run on mobile phones. If a mobile user moves to another network then the NMS is migrated in order to achieve better performance. In order to test the Service Management Framework, the NMS will be migrated from one cloud server to another. In this paper, we show how to implement the NMS in the Docker’s private repository and migrate it to a new location. We installed Docker on a Linux Ubuntu 20.04 machine and set up a private repository to store our services. This allows us to use computing resources for migrating to cloud environments.

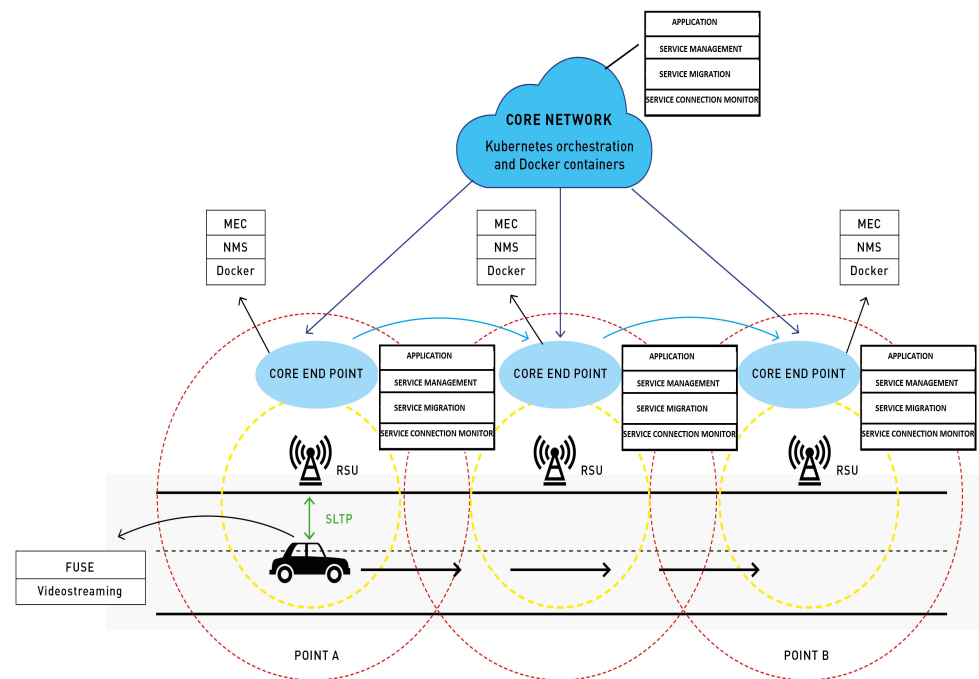


Figure 11. Implementation scenario: FUSE architecture with NMS.

7.6. Requirements Table

Table 6 explains the requirements for the prototype implementation to do a successful migration.

Table 6. Requirements table for Prototype.

Name	Details
Docker private repository	Secure Migration
NMS	Microservices
RASP	Migration Protocol
SMF	Service Management Framework
Fuse Client	Uses NMS as Backing Store
Capabilities	Users, Devices, Service, Servers

7.7. Make Fuxfs Multithreaded

In order to develop the prototype, it was decided to make the client (i.e., the fuxfs program) multithreaded. One thread called SMF_THREAD was used to manage the interaction with the SMF as shown in Figure 12. In this prototype, the FUSE client was used to tell the SMF to migrate the service.

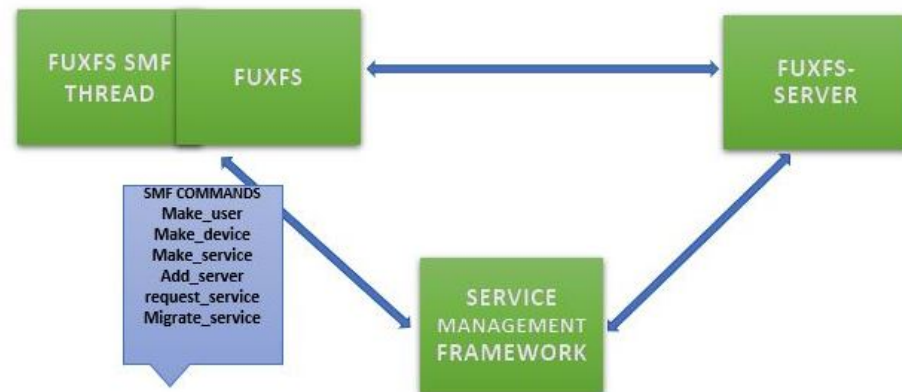


Figure 12. Fuxfs Multithread.

7.8. SMF Commands

The SMF was developed to manage services and migrate and replicate services in a secure environment. To check secure migration on the Service Management Framework (SMF), we developed C programs on the Ubuntu 20.04 platform. We used a private Docker [18] repository to migrate our service and tested them in our local environment. The SMF provides the following functions:

- **Register Service:** The function allows the service to be registered with the SMF. It collects information on the service, including the service name, the description of the service, the version of the service (i.e., 0 is the latest version), the TCP/UDP port number, and the service capability. Once the service has been successfully registered with the SMF, the SMF generates the global_id. The local_id is generated by the CMS.
- **Register User:** This function helps the new user register with the SMF. It collects information about the user, including the first name, surname, role, rank, and specialisation, as well as the capability of the user to register. Once the user has been successfully registered with the SMF, the SMF will generate their global user ID.
- **Register Device:** This register device function allows the devices to register with SMF. To register a device, the name of the device, the first name and surname of the device owner, and the device capability are also collected once the device is registered. Each device will have a unique local device ID and a global device ID.
- **Add Server:** This function helps a server be added to the required service. It also collects details about the server, including the server global ID given by the NMS, the server's name, the server location using its IPv4 address, and the server's maximum load.
- **Request Service:** Once a service is registered with the SMF, the service can be requested by users who want to use it. Servers should provide this service to the requesting applications. This function allows an application to request service. After the application sends a service request to SManL, SManL will provide the necessary parameters for the application to contact the server that runs the requested service.
- **Migrate Service:** This function allows the services to migrate to the required servers. It also collects details about the global service ID, the global user ID, and the global device ID and will set up the migration using Docker container technology. Once the migration is successful, a successful result will be returned to the caller.

7.9. FUSE Server Migration Using Docker

In the source code compilation, using the command prompt, compile the Fuse server (fuxfs_server.c). The next step is to run Docker, build the Docker image and check the built image. To secure the service migration, we have created a private repository in Docker for the migration. As we know that the SMF supports the following functions: Register the service, register the user, register the device, add a server, request and migrate the service. It was compiled and executed using the below commands to start the functions. Figure A2 in the Appendix A shows that the SMF has successfully started.

- Compile command: `gcc -o smf smf.c`
- Run command: `./smf`

7.9.1. Fuxfs Server Start-Up from Dockers Private Repository

After the SMF started, the Source code of “Fuxfs_server.c” compiled and was executed as shown in Figure A3 in the Appendix A, which highlights that the Fuxfs server was started from the Docker Repository.

- Compile command: `gcc -o Fuxfs server Fuxfs server.c`
- Run command: `./Fuxfs server 0.0.0.0`
- docker run command: `gayuinfy/gkprivate:v1`

This Fuxfs server derives from Docker private container, we used the docker run command: `gayuinfy/gkprivate:v1` and the container ID: `ff0fa32d3036` as shown in Figure A4 in the Appendix A.

7.9.2. Fuxfs Client Start-Up

After the Fuse server, the fuse client (Fuxfs.c) was compiled and executed as shown in Figure A5 in the Appendix A.

- Compile command : `gcc -Wall fuxfs.c 'pkg-config fuse --cflags --libs' -o fuxfs -w -lpthread`
- Run command: `./fuxfs /tmp/fuse 0.0.0.0`

7.9.3. NMS Service Added

The NMS is an example of block storage service, registered with the SMF. The “Add server” command from SMF allows servers to be added including Server name, type, Server IP address, TCP and UDP port details has shown in Figure A6 in the Appendix A.

Once the service was successfully added to the server, the service was requested by the FUSE client using the SMF command Request service. The FUSE client then requested that the NMS was migrated to a different machine. The service is then successfully migrated as shown in Figure A7 in the Appendix A. Figure A8 in the Appendix A shows all the components of the working prototype.

8. Conclusions and Future Work

This research has focused on the development of the Secure Service Ecosystem (SSE) to implement the Intelligent Edge Environment for smart cities. This effort looked at three layers of the IEE and developed key mechanisms to help implement the SSE including a new Resource Allocation Algorithm, a new Resource Allocation Secure Protocol and finally a new Secure Service Protocol. Using these mechanisms along with capabilities, ProVerif was used to show that this system was safe. The SSE was then used to support a FUSE-NMS system in which the NMS was migrated as the users moved around.

Though the prototype demonstrated was basic, it showed that it was possible to build an SSE that could be used to develop new services for smart cities. We therefore believe that this approach will allow services for smart cities to be developed and deployed in a much shorter time frame as these services will be automatically managed by the Service Management Framework not by the developers of the service.

For future work, we need to explore how we use this architecture to support a number of microservices for the vehicular environment including vehicle health monitoring and remote navigation and control systems.

Author Contributions: Formal analysis, algorithms, and software implementation, G.K.; protocols, BCSL, and software implementation, G.M. writing, review, and editing, G.K. and G.M. All authors have read and agreed to the published version of the manuscript.

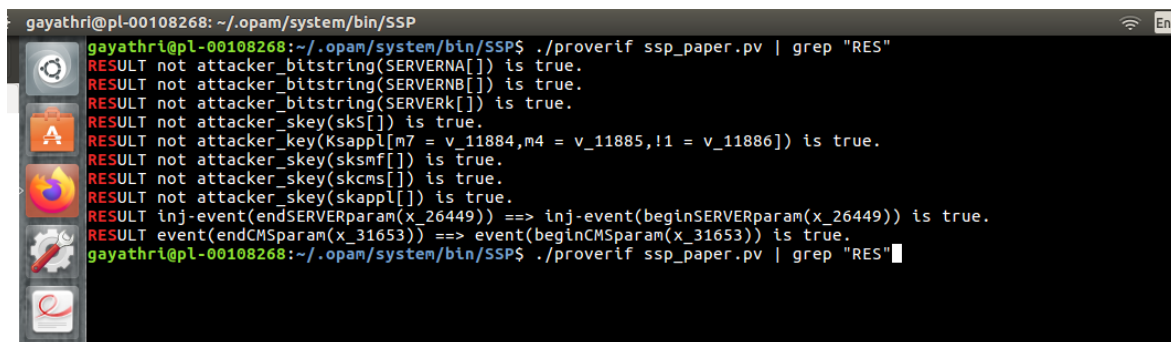
Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding authors.

Acknowledgments: The authors would like to thank Florian Kammüller and Mahdi Aiash for their help and guidance during this effort. We would also like to thank the Department for Transport (DfT) for funding the Middlesex Connected Vehicle testbed which was the motivational starting point for this research.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

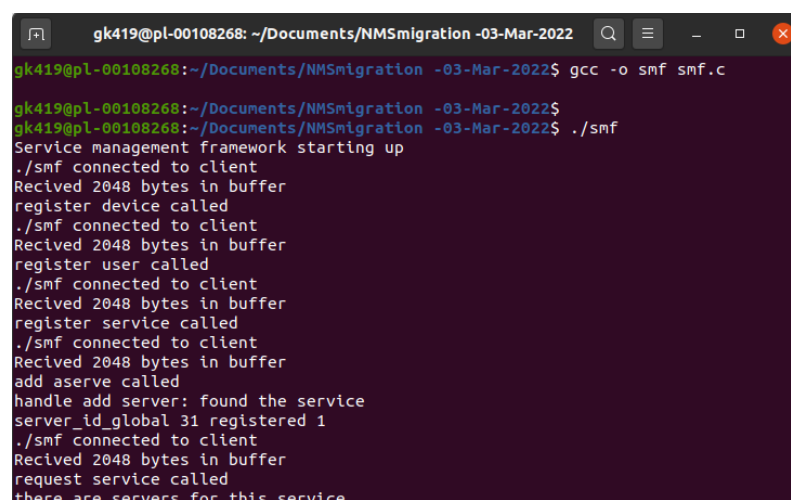


```

gayathri@pl-00108268: ~/opam/system/bin/SSP
gayathri@pl-00108268:~/opam/system/bin/SSP$ ./proverif ssp_paper.pv | grep "RES"
RESULT not attacker_bitstring(SERVERNA[]) is true.
RESULT not attacker_bitstring(SERVERNB[]) is true.
RESULT not attacker_bitstring(SERVERK[]) is true.
RESULT not attacker_skey(sks[]) is true.
RESULT not attacker_key(Ksappl[m7 = v_11884,m4 = v_11885,!1 = v_11886]) is true.
RESULT not attacker_skey(sksmf[]) is true.
RESULT not attacker_skey(skcms[]) is true.
RESULT not attacker_skey(skappl[]) is true.
RESULT inj-event(endSERVERparam(x_26449)) ==> inj-event(beginSERVERparam(x_26449)) is true.
RESULT event(endCMSparam(x_31653)) ==> event(beginCMSparam(x_31653)) is true.
gayathri@pl-00108268:~/opam/system/bin/SSP$ ./proverif ssp_paper.pv | grep "RES"

```

Figure A1. SSP protocols results.



```

gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ gcc -o smf smf.c
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ ./smf
Service management framework starting up
./smf connected to client
Received 2048 bytes in buffer
register device called
./smf connected to client
Received 2048 bytes in buffer
register user called
./smf connected to client
Received 2048 bytes in buffer
register service called
./smf connected to client
Received 2048 bytes in buffer
add aserve called
handle add server: found the service
server_id_global 31 registered 1
./smf connected to client
Received 2048 bytes in buffer
request service called
there are servers for this service

```

Figure A2. SMF running successfully.

```

gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ ./fuxfs /tmp/fuse 0.0.0.0

Allocating block from the server
Connected to the server 0.0.0.0
I've sent 8197 bytes over the network
I've received 4 bytes from the network
Received block number 1
Time spent in allocate_block is 380 us
Add entry . to inode 1

Writing block number 1 to server
Connected to the server 0.0.0.0
I've sent 8197 bytes over the network
Time spent in write_block is 107 us
Add entry .. to inode 1

Writing block number 1 to server
Connected to the server 0.0.0.0
I've sent 8197 bytes over the network
Time spent in write_block is 94 us
Add entry lost+found to inode 1

Writing block number 1 to server

```

Figure A3. Fuxfs server start-up.

```

gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
45ee310447d7 postgres:10.10 "docker-entrypoint.s..." 2 weeks ago
  Exited (0) 2 weeks ago      objective_lichterman
d84aa793618a postgres:9.6 "docker-entrypoint.s..." 2 weeks ago
  Exited (1) 2 weeks ago     cool_jackson
3f5b05308f38 gayuinfy/myfuse "0.0.0.0" 2 weeks ago
  Created                    youthful_faraday
6912607cce21 gayuinfy/myfuse "/bin/sh -C /home/fu..." 2 weeks ago
  Exited (255) 2 weeks ago   suspicious_meninsky
154f5d0a39aa ubuntu "bash" 2 weeks ago
  Exited (0) 2 weeks ago     kind_goodall
e754cfbce357 myprg-gayu ".myprg" 2 weeks ago
  Exited (0) 2 weeks ago     confident_greider
1e0daa82deaf ubuntu "bash" 3 weeks ago
  Exited (127) 3 weeks ago   trusting_lovelace
13c9eadf7e10 hello-world "/hello" 3 weeks ago
  Exited (0) 3 weeks ago     zen_neumann
c1e67afba25e hello-world "/hello" 3 weeks ago
  Exited (0) 3 weeks ago     intelligent_chatelet
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ docker ps
CONTAINER ID   IMAGE          NAMES                COMMAND                CREATED        ST
ATUS          PORTS         NAMES                COMMAND                CREATED        ST
ATUS
ff0fa32d3036  gayuinfy/gkprivate:v2  "/app/fuxfs_server"  34 seconds ago       Up
33 seconds   1068/tcp     thirsty_ellis
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$

```

Figure A4. Fuxfs server start-up from Docker's private repository.

```

gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ ./fuxfs_server 0.0.0.0

I've received 8197 bytes from the network
buffer offset is 8197

Allocate block
I've allocated block number 1
I've sent 4 bytes over the network

I've received 0 bytes from the network
Socket was closed

I've received 8197 bytes from the network
buffer offset is 8197

Writing block 1

I've received 0 bytes from the network
Socket was closed

I've received 8197 bytes from the network
buffer offset is 8197

Writing block 1

```

Figure A5. Fuxfs client start-up.

```

gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
made service
Adding server
adding server to a service
Please type service name: press enter when finished
NMS
NMS
Is this correct: Type y for YES and n for NO: press enter when finished
y
service match found
add_server: found service:NMS
adding server to service: NMS
Please type name of the server: Type na if not applicable: press enter when finished
TG09
TG09
Is this correct: Type y for YES and n for NO: press enter when finished
y
add_server: please type the server location in terms of the IPv4 address in dot form: fo
r example 192.15.4.0. Press enter when finished
192.168.0.1
192.168.0.1
Is this correct? Type y for YES and n for NO:

```

Figure A6. A Server is added to the NMS Service.

```

gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
Request Service starting up
connected to the server
Please type name of service: press enter when finished
NMS
NMS
Is this correct: Type y for YES and n for NO: press enter when finished
y
Result returned: 2048
Service Registered:Global service id :11
Global server id: 31

Setting up migration
Migration set up and Ready to go
Migrating the Service
Migrate Service starting up
connected to the server
Result returned: 2048
Service Registered:Global service id :11
Successfully Migrated the Service
Exiting..

I'm in lookup!

```

Figure A7. NMS service migrated.

The figure consists of four terminal windows:

- Top Left:** Shows the execution of `smf` commands, including `smf connect to client`, `smf register user called`, `smf register service called`, `smf add server: found the service`, `smf request service called`, `smf migrate`, and `smf fuse server starting up`.
- Bottom Left:** Shows the execution of `add_server` and `request service` commands, with output indicating the server is added and the service is requested.
- Top Right:** Shows the execution of `smf migrate` and `smf fuse server starting up` commands, with output indicating the migration is successful and the service is running.
- Bottom Right:** Shows the output of `docker ps`, displaying a list of containers including `docker-entrypoint-s...`, `objective_lichterman`, `cool_jackson`, `youthful_faraday`, `kind_goodall`, `confident_grelder`, `zen_neumann`, and `Intelligent_chatelet`.

Figure A8. SMF (top left), FUSE Client (bottom left), NMS (top right), and Docker Status (bottom right): NMS running at a different location using the SMF.

References

1. Gayathri, K.; Glenford, M.; Jon, C. Building an Intelligent Edge Environment to Provide Essential Services for Smart Cities. In Proceedings of the MobiArch '23, Madrid, Spain, 6 October 2023; pp. 13–18. [CrossRef]
2. Pavel, M.; Zdenek, B. Mobile edge computing: A survey on architecture and computational offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
3. Edgeless: Cognitive Edge-Cloud with Serverless Computing. Available online: <https://edgeless-project.eu/> (accessed on 18 July 2024).
4. CODECO: A Novel Edge-Cloud Orchestration Framework, Focusing on Data-Compute-Network. Available online: <https://he-codeco.eu/> (accessed on 18 July 2024).
5. Paranthaman, V.V.; Ghosh, A.; Mapp, G.; Inioiosa, V.; Shah, P.; Nguyen, H.X.; Gemikonakli, O.; Rahman, S. Building a Prototype VANET Testbed to Explore Communication Dynamics in Highly Mobile Environments. In *Testbeds and Research Infrastructures for the Development of Networks and Communities. TridentCom 2016*; Guo, S., Wei, G., Xiang, Y., Lin, X., Lorenz, P., Eds.; Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering; Springer: Cham, Switzerland, 2017; Volume 177. [CrossRef]
6. Xiaolong, X.; Bowen, S.; Xiaochun, Y.; Mohammad, R.; Khosravi, H. Edge Server Quantification and Placement for Offloading Social Media Services in Industrial Cognitive IoV. *IEEE Trans. Ind. Inform.* **2021**, *17*, 2910–2918. [CrossRef]
7. Onyekachukwu, A.E.; Gayathri, K.; Glenford, M.; Ramona, T. Exploring the Provision of Reliable Network Storage in Highly Mobile Environments. In Proceedings of the 2020 13th International Conference on Communications (COMM), Bucharest, Romania, 18–20 June 2020; pp. 255–260. [CrossRef]
8. Chen, S.; Zhou, M. Evolving container to unikernel for edge computing and applications in process industry. *Processes* **2021**, *9*, 351. [CrossRef]
9. Kifayat, U.; Luz, M.S.; Joao, B.R.; Edson, D.M. A lightweight beaconing-based commercial services advertisement protocol for vehicular ad hoc network. In Proceedings of the International Conference on Ad-Hoc Networks and Wireless, Lille, France, 4–6 July 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 279–293.
10. Jennings, B.; Stadler, R. Resource management in clouds: Survey and research challenges. *J. Netw. Syst. Manag.* **2015**, *23*, 567–619. [CrossRef]
11. Karthick, G.; Mapp, G.; Kammüller, F.; Aiash, M. Formalization and Analysis of a Resource Allocation Security Protocol for Secure Service Migration. In Proceedings of the IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, Switzerland, 17–20 December 2018; pp. 207–212. [CrossRef]
12. ProVerif: Cryptographic Protocol Verifier in the Formal Model. Available online: <https://bblanche.gitlabpages.inria.fr/proverif/> (accessed on 18 July 2024).
13. Blanchet, B. Modeling and Verifying Security Protocols with the Applied pi Calculus and Proverif. *Found. Trends® Priv. Secur.* **2016**, *1*, 148. Available online: <https://ieeexplore.ieee.org/document/8186937> (accessed on 18 July 2024).
14. Mapp, G.; Aiash, M.; Guardia, H.C.; Crowcroft, J. Exploring Multi-homing Issues in Heterogeneous Environments. In Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications, Singapore, 22–25 March 2011; pp. 690–695. [CrossRef]
15. Vithanwattana, N.; Karthick, G.; Mapp, G.; George, C.; Samuels, A. Securing future healthcare environments in a post-COVID-19 world: Moving from frameworks to prototypes. *J. Reliab. Intell. Environ.* **2022**, *8*, 299–315. [CrossRef] [PubMed] [PubMed Central]
16. Jose, R.; Onyekachukwu, A.E.; Gayathri, K.; Ramona, T.; Glenford, M. A New Service Management Framework for Vehicular Networks. In Proceedings of the 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 24–27 February 2020; pp. 162–164. [CrossRef]
17. Karthick, G.; Mapp, G.; Kammüller, F.; Aiash, M. Modeling and verifying a resource allocation algorithm for secure service migration for commercial cloud systems. *Comput. Intell.* **2022**, *38*, 811–828. [CrossRef]
18. Docker: Containerize Your Applications. Available online: <https://www.docker.com/> (accessed on 18 July 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.