A trust evaluation scheme of service providers in mobile edge computing

Merrihan Badr Monir Mansour[1,2], Tamer Abdelkader[1], Mohammed Hashem AbdelAziz[1], El-Sayed Mohamed EI-Horbaty[1]
[1]Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt
[2]Faculty of Business Administration Economics and Political Science, The British University in Egypt, Cairo, Egypt

Article Info

Article history:

Received Dec 25, 2020
Revised Sep 19, 2021
Accepted Oct 10, 2021

Keywords:

Edge computing
Processing performance
Processing throughput
Service level agreement
Service providers
Trust evaluation

ABSTRACT

Mobile edge computing (MEC) is a new computing paradigm that brings cloud services to the network edge. Despite its great need in terms of computational services in daily life, service users may have several concerns while selecting a suitable service provider to fulfil their computational requirements. Such concerns are: with whom they are dealing with, where will their private data migrate to, service provider processing performance quality. Therefore, this paper presents a trust evaluation scheme that evaluates the processing performance of a service provider in the MEC environment. Processing performance of service providers is evaluated in terms of average processing success rate and processing throughput, thus allocating a service provider in a relevant trust status. Service provider processing incompliance and user termination ratio are also computed during provider's interactions with users. This is in an attempt to help future service users to be acknowledged of service provider's past interactions prior dealing with it. Thus, eliminating the probability of existing compromised service providers and raising the security and success of future interactions between service providers and users. Simulations results show service providers processing performance degree, processing incompliance and user termination ratio. A service provider is allocated to a trust status according to the evaluated processing performance trust degree.

Corresponding Author:

Merrihan Badr Monir Mansour
Faculty of Computer and Information Sciences, Ain Shams University
El-Khalifa El-Maamoun, Al Obour, Al Qalyubia Governorate, Egypt
Email: merrihan.mansour@bue.edu.eg

1. INTRODUCTION

Mobile edge computing (MEC) is a new emerging technology that extends the cloud computing capabilities to the network edge [1], by integrating MEC servers with the mobile network edge [2], through radio access network (RAN) [3], [4]. This permits direct mobile communication between the base network and end users [5], which allows low latency, better quality of service (QoS) [6], high bandwidth access to mobile applications and network information [7]. With the great evolution of mobile devices' capabilities, their owners hold valuable information, apart from the devices' configuration, such as real time knowledge and on-time location awareness of an event. Such mobile capabilities and information are considered great resources in terms of data analysis, processing, and storage media [8]. With the MEC network expansion [9], there is a great increase in service providers offering services. In this context, there could be different service providers offering similar service types, e.g., processing computation and/or storage, were each of them could have different processing performance quality. On the other hand, one service provider could offer more than one service type [10]. However, not all services offered by the same service provider could have the same processing performance efficiency. Meanwhile, service users could require different functionalities and have different processing preferences, in terms of cost, storage capacity, processing performance quality and trust degree [11], [12].

Given that, service providers are located in remote locations, most of them are unknown to service users. For this reason, service users hold several doubts such as, dealing with unknown service providers, user's data privacy, service providers' history and processing performance quality. This is mainly due to the lack of previous experience between service providers and users. This creates a level of uncertainty about the fulfilment of service users' various computational needs and expectations, which limits users' dependency on the MEC resources [13]. Many researchers had presented various attempts to build trusted relationships in edge computing paradigms [14]. This is to provide an efficient trust evaluation scheme for the available services provided by service providers, to secure future users' interactions [15], [16]. In mobile edge computing, a secure multi-tier model was proposed in [17]. In this protocol, it was assumed that the higher degree of trust, the less security measures could be taken by a node and vice versa. But unfortunately, it did not consider sudden attacks occurring for a trusted node, such as hacking. A trust evaluation scheme was presented in [18], were it computes service providers' identity, hardware capabilities and behavioral trust in the MEC network. The main limitation of this scheme is that it mainly depends on users' feedback opinion to compute trust.

An integrated trust evaluation model was depicted in [19], to evaluate service providers' identity, historical behavior and quality of service offered. Trust computation was time consuming in this model, due to the complexity of the equations. In [20], a trust assessment protocol was developed that monitors and analyze traffic flow of interactions between service users and providers to evaluate trust. However, no performance parameters were evaluated. Another attempt to evaluate service providers' performance during their interactions is the issuance of a service level agreement (SLA) [21]. An SLA is an agreed upon document between a service provider and user, that specifies the required task description and application requirements [22]. Yet, this is not sufficient to secure service users, since not all service providers abide to the SLA statements thoroughly. However, there is a lack of a standard SLA format. As shown, each of the previously mentioned protocols measured cloud services using different parameters. There is not a unified scheme that could evaluate service providers processing performance. Meanwhile, node history was not captured, which gives a chance for an entity to behave maliciously, knowing that it would not be recognized in the future. This increases service users' fears and prevents them from relying on the cloud and edge computing paradigms to fulfil their computational needs.

This paper presents a unified trustworthy evaluation model that evaluates service providers' processing performance, to distinguish trustworthy providers. The main contributions of this paper are: i) service providers' processing performance evaluation in terms of processing success ratio and throughput, ii) processing incompliance and user termination ratio computation of service providers, iii) development of a penalty system to track malicious actions committed by service providers, and iv) assignment of service providers to relative trust status. This would help service users in their service providers' selection and optimizes the security of future interactions, which enhances the MEC network expansion [23].

The rest of this paper is organized as follows; section 2, introduces the proposed architecture method, functional algorithms, and their description. Section 3 shows the results and discussion. Finally, the conclusion and future work are presented in section 4.

2. PROPOSED ARCHITECTURE METHOD

The proposed architecture evaluates the processing performance of service providers in the MEC network. In this model, a service provider is evaluated according to its processing performance quality and not by the quantity of hardware or software resources that it possesses, e.g., storage space, number of processors and RAM. The main protocol entities, proposed functions, equations, and algorithms are described in subsection 2.1 to 2.6.

2.1. Main protocol entities and their equivalent tasks

The main acting entities in the proposed scheme are service provider SP, service user SU, cloud broker (CB), network provider (NP) and cloud service manager (CSM) [24]. The relationship between the protocol entities is shown in Figure 1 and detailed below.

- Service provider SP_i: [25] a service provider "i", i Є I, where I is the set of service providers. A service provider may be a small entity offering one service of one type, or a big organization that owns several hardware and software resources and offers several services of different job types. In case a service

provider SP_i provides more than one service type, it is referred to as Sr_i. For simplicity, we will be referring to each service provider as Sr_i, throughout this paper.
- Service user SU_j: a service user "j", j Є J, where J is the set of service users. A service user could be an entity or organization that requests a specific service to be performed over the network and pays for it [26].
- Cloud broker CB_u: a cloud broker "u", u Є U, where U is the set of cloud brokers. CB is an entity that mainly helps service users to find appropriate service providers to fulfil their computational needs, and it is being paid for this job. CB works as a local entity per area, where it should be aware of all available service providers and their offered service type. CB could communicate with other entities outside its area to reach suitable service providers [22]. CB is considered as a semi trusted entity that is not allowed to reveal service provider or service user private information, such as own opinion, as it could have a personal benefit. It also acts as a transmission and storage medium between service provider, user and CSM for certain encrypted information as shown below.
- Cloud service manager (CSM): is considered a fully trusted authorized entity that is responsible for registering cloud brokers, service providers and service users to the MEC network through a network provider. CSM evaluates the processing performance and trust status of service providers [27]. It also checks the status and validity of cloud brokers periodically to ensure secure communication medium between service providers and users. Therefore, CSM should maintain high computational capabilities and covers a wide geographical region.
- Network provider NP_w: a network provider "w", w Є W, where W is the set of network providers. NP is responsible for communication, data transmission and network efficiency, between all the above entities. Note that, there could be more than one network provider located per geographic area [12].

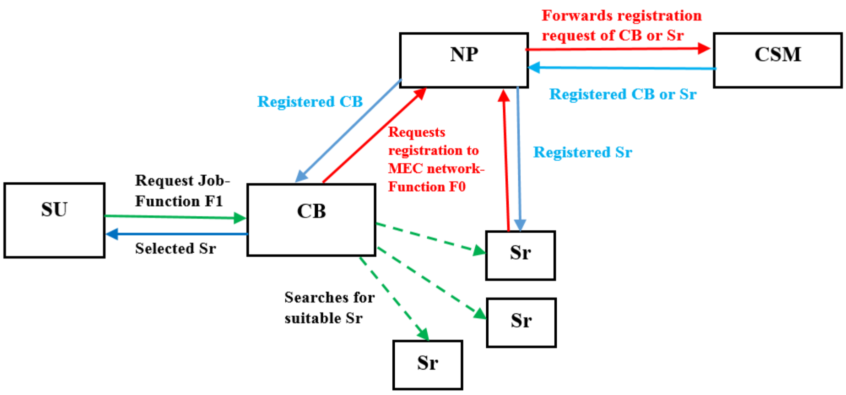

Figure 1. Main protocol entities relationship

However, a cloud broker, service provider or service user could deal with more than one network provider [28]. While a service provider could accept jobs from more than one cloud broker, a service user can also deal with more than one cloud broker to request different computational tasks. All of the above entities are authenticated in the MEC network by their unique identity, which is out of the scope of this paper.

2.2. List of assumptions

The proposed scheme considers the following assumptions:

a) There are three job types requested over the MEC network; type 1: storage request (storing massive terabytes, e.g., videos), type 2: computational processing request (jobs that require high processing speed/capabilities), type 3: requesting both of them. These jobs are the most commonly requested processes in the cloud computing paradigm and MEC network;
b) If the same service provider SP_i offers more than one job type, known as Sr_i, were r Є {type 1, type 2, type 3}. This does not mean that SP has the same computational efficiency for all job types. For instance, it could be powerful in one job type, e.g., storage, and weak in another, e.g., processing efficiency or vice versa;
c) The same service provider SP_i gives equal usage and benefits of its hardware and software resources to all its services and users;
d) Job processing, storage and execution isolation is considered as a default action by a service provider [29];

e) Since each service user could have different priorities and intentions, the proposed scheme asks the service user for a priority list of preferences and responds with a recommendation list accordingly;
f) Batch processing is assumed for trust computation. All processes of the same job type per service provider Sr_i are gathered per computational interval (e.g., month);
g) Trust evaluation is performed per process accepted by Sr_i; 8) A network provider of each of Sr_i, SU_j and CB_u, have a limited contribution in the proposed scheme as discussed below.

2.3. Processing performance computational equations

The proposed trust scheme measures the processing performance P(Sr_i) of service provider SP_i which could own more than one service Sr_i. However, jobs of the same type, are evaluated per service provider Sr_i, as mentioned previously. Noting that, P consists of different weighted parameters and computed by the CSM, as described below.

Let each requested job of any type be known as process "a", a Є A, where A is the total number of processes executed at Sr_i, but distinguished by their job type. Each process "a" has a separate service level agreement, even if it is performed by the same SP_i/Sr_i. Assume the components of an SLA of process "a" executed by Sr_i be processing cost (SC_{ia}), storage capacity GB/TB (SS_{ia}), duration of maintenance hr/min (SM_{ia}), and agreed estimated execution time hr/min (SE_{ia}) [30]. All SLA's components are rated by SU_j after the job is ended or terminated [31]. The proposed scheme constitutes of eight equations, as detailed below.

Let T_v be actual processing execution time of process "a" at Sr_i, where "a" start time is Pa_{st} and end time is Pa_{et}. Hence,

$$T_v = Pa_{et} - Pa_{st} \tag{1}$$

Assume the time difference between the estimated agreed time SE_{ia} and actual processing execution time T_v, be TR_{ia} (Time compliance), therefore,

$$TR_{ia} = SE_{ia} - T_v \tag{2}$$

$$TR_{ia} = \begin{bmatrix} \geq 0 & \text{"}a\text{" completed within } \mathrm{SE_{ia}} \\ < 0 & \mathrm{SE_{ia}} \text{ time incompliance} \end{bmatrix}$$

Given that each process "a" should end in one of the following four states $\{\mathrm{P_{E1}}, \mathrm{P_{E2}}, \mathrm{P_{T1}}, \mathrm{P_{T2}}\}$:

$$\begin{bmatrix} \mathrm{P_{E1}} & \text{Process ended by } \mathrm{Sr_i} & \text{complete} \\ \mathrm{P_{E2}} & \text{Process ended by } \mathrm{Sr_i} & \text{incomplete} \\ & & \end{bmatrix}$$

$$\begin{bmatrix} \mathrm{P_{T1}} & \text{Process terminated by } \mathrm{SU_j} & \mathrm{T_v} > \mathrm{SE_{ia}} \\ \mathrm{P_{T2}} & \text{Process terminated by } \mathrm{SU_j} & \mathrm{T_v} < \mathrm{SE_{ia}} \\ & & \end{bmatrix}$$

Let $P_{E1\_}T$, $P_{E2\_}T$, $P_{T1\_}T$ and $P_{T2\_}T$ be the total number of P_{E1}, P_{E2}, P_{T1} and P_{T2} respectively. Let the rated SLA, be $SLA_{ia\_}R$, and the total number of $SLA_{ia\_}R$ implies the total number of ended jobs/processes, "A". Therefore, the average processing success rate $P_{AV}\,(Sr_i)$ can be measured by,

$$P_{AV}\,(Sr_i) = \frac{P_{E1\_}T}{A} \tag{3}$$

On the other hand, processing incompliance $PI(Sr_i)$ or failure ratio, can be calculated as,

$$PI(Sr_i) = \frac{P_{E2\_}T + P_{T1\_}T}{A} \tag{4}$$

The user termination ratio $UTR(Sr_i)$ can be measured by,

$$UTR(Sr_i) = \frac{P_{T2\_}T}{A} \tag{5}$$

In case $UTR(Sr_i)$ exceeds a certain threshold, a warning is issued to alarm the relevant service provider of its high user termination ratio. The processing throughput PT($\boldsymbol{Sr_i}$) will be measured by considering the number of completed successful processes P_{E1}, per computational interval and given certain points for each range of values by,

$$\text{PT}(Sr_i) = \text{Points}(Sr_i) \tag{6}$$

for $1 \leq P_{E1}\_T$ // Function F3.

After all, the processing performance of a service provider $\text{P}(\boldsymbol{Sr_i})$ will be computed as,

$$\text{P}(Sr_i) = \frac{P_{AV}\,(Sr_i) + \text{PT}(Sr_i)}{2} \tag{7}$$

Noting that the total number of processes executed, "A", at service provider Sr_i of SP_i, per computational interval, equals;

$$A = P_{E1}\_T + P_{E2}\_\text{T} + P_{T1}\_T + P_{T2}\_T \tag{8}$$

as process "a", should end in one of these four states. The processing capacity PC_i of a service provider Sr_i, is a predetermined value by the service provider specified during the registration process. Each of the above equations is applied in the below functional algorithms, as part of the trust computation process.

2.4. Functional algorithms and description

The proposed protocol is composed of eleven functional algorithms, which leads to measuring the processing performance of a service provider Sr_i and assigning it to a trust status. It also measures the processing incompliance and user termination ratio per service provider. A penalty system is also presented to identify any malicious action performed by the participating entities. Table 1 shows each function name and aim, the consequent action performed with the relevant algorithm figure number. Figure 2 presents the protocol architecture and the relationship between the eleven functions. Dashed lines indicate that this function may or may not be called.

Table 1. Proposed scheme functions

Function name	Aim	Action performed	Algorithm no.
F0	Service provider registration function.	Service provider registered to MEC network if it is not previously registered in network.	1
F1	Job request and execution protocol.	Job processed by service provider.	2
F2	Cloud broker computation of total SLA_{ia}_R, "A" for each service provider Sr_i.	"A" computed per service provider Sr_i.	3
F3	Processing throughout computation for each Sr_i.	$\text{PT}(Sr_i)$ computed.	10
F4	Processing performance evaluation for Sr_i.	$\text{P}(\text{Sr}_\text{i})$, $\text{PI}(\text{Sr}_\text{i})$, Sr_i_age, $\text{UTR}(\text{Sr}_\text{i})$ & $P_{AV}\,(Sr_i)$ computed.	9
F5	Trust status computation function for Sr_i.	$\text{T}_\text{n}(\text{Sr}_\text{i})$ computed.	11
F6	Cloud broker checks SLA_{ia}_R validity.	Penalty E1_SU_j is set to true or false accordingly.	4
F7	Complain function against service provider Sr_i.	Cloud broker stops sending new job_request to $\boldsymbol{Sr_i}$ until TR_{ia} correction is sent by it.	5
		Penalty Functions	
E1	SU_j did not send SLA_{ia}_R.	All job requests by SU_j are rejected until it sends SLA_{ia}_R.	6
E2	Sr_i attempt to register again as a new provider in the network.	$\text{P}(\text{Sr}_\text{i})$ is decreased in the first attempt. While Sr_i is temporarily blocked from accessing the network for X-days in the later attempts by penalty "E5".	7
E3	Lazy or compromised cloud broker.	Generates W1 or W2 in the first and second attempts. While CB_u is temporarily blocked from accessing the network for X-days by penalty "E4".	8

2.5. Proposed penalty protocol

The penalty protocol is presented as part of the proposed architecture. It mainly aims to track service providers malicious actions, which helps in their processing performance and trust status computation process. However, the penalty protocol achieves the following: i) it shows the type of wrong action performed by the involved entity; ii) it encounters several malicious action types expected to happen, such as an existing service provider attempt to register as a new one, in order to hide its bad history; iii) it counts the number and type of malicious actions performed per computational interval; and iv) imposes a penalty according to each malicious action type performed by the accused entity.

The penalty scheme also monitors cloud brokers and service users' actions in a limited manner, since this is out of the scope of this paper. This provides more secure transactions between service providers, users, and cloud brokers in the MEC environment. Table 2 states each penalty name, its description, and consequences. All of the above functions are described in the following subsections with their relevant algorithms.

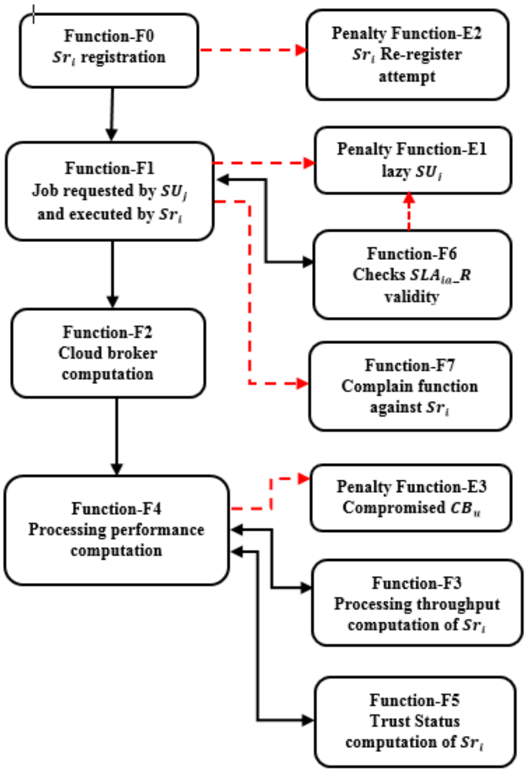

Figure 2. Functions relationship diagram

Table 2. Penalties and their consequences

Penalty name	Description	Consequences
E1_SU_i	SU_i did not send to CB_u rated(SLA_{ia}_R) after process ended/terminated.	Job request rejected until sending the pending SLA_{ia}_R.
E2_Sr_i	Sr_i first attempt to register again in the MEC network.	Request rejected, $P(Sr_i)$ decreased.
E3_CB_u	Cloud broker CB_u exceeded end of month batch computation for Sr_i for one or two attempts.	Cloud broker receives warning W1 or W2 from CSM accordingly.
E4_CB_u	Third attempt of cloud broker to delay its computational job, after being warned by "E3". Compromised CB_u.	Cloud broker is temporarily blocked from the network by CSM.
E5_Sr_i	Sr_i second attempt to register again in the MEC network, after being warned by "E2". Compromised Sr_i.	Service provider is temporarily blocked from the network by CSM, for X-days.

2.6. Functional algorithms

In this section, each of the above mentioned functions in Table 1, are described below with their relative algorithms. The penalty functional algorithms are also detailed below, together with their generated warnings, and actions performed accordingly.

2.6.1. Service provider registration

A service provider SP_i could own more than one service, thus service registration to the MEC network is performed per service Sr_i {job type 1, 2, or 3} of service provider SP_i, with the aid of a network provider. Algorithm1, shows the service provider registration algorithm 1, function F0, and its flowchart in Figure 3.

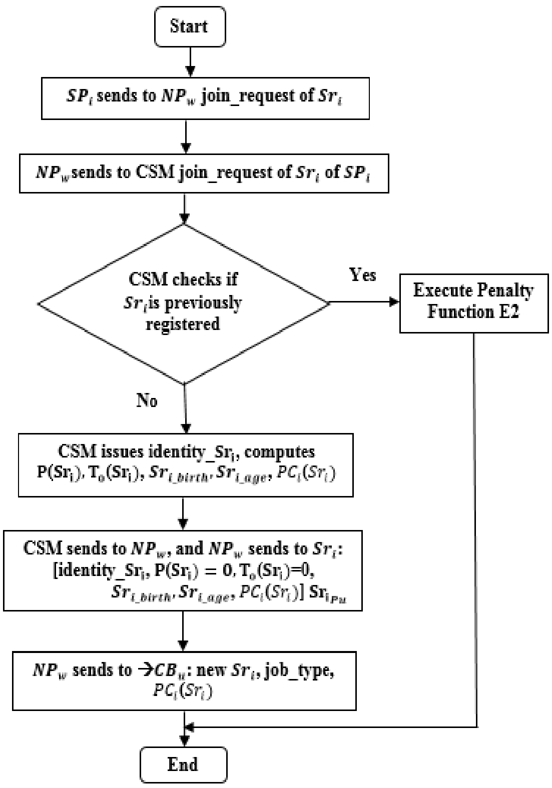

Figure 3. Service provider registration flowchart

Algorithm 1: Service provider registration-function F0

Algorithm code: F0
Description: Service provider registration function.
Executed at CSM.

Input: new Sr_i of SP_i needs to register to MEC network.
Output: registered Sr_i of SP_i.
SP_i→NP_w: join_request of Sr_i of SP_i
NP_w→CSM: join_request of Sr_i of SP_i ,
where join_request *includes* job_type = (storage, processing, or both & PC_i_value)
CSM: *checks*
    if Sr_i exists *then*
        execute Penalty Function E2
        exit ()
    else
      CSM:
       issues identity_Sr_i // Sr_i unique credentials
       computes
       P(Sr_i)=0, T_o(Sr_i)= "Beginner", // T_o(Sr_i)= initial trust status of Sr_i
       assigns $PC_i(Sr_i)$=value
       Sr_{i_birth} = Current_date, // birth date= registration day
       Sr_{i_age}= 0 // initial state // age of Sr_i
      CSM→NP_w:
       NP_w→Sr_i: (E_n(identity_Sr_i, P(Sr_i) = 0, T_o(Sr_i)=
          "Beginner", Sr_{i_birth},Sr_{i_age}, $PC_i(Sr_i)$) $SP_{i_{Pu}}$)
                //encrypted by the public key of SP_i
       NP_w→CB_u: new Sr_i+ job_type+$PC_i(Sr_i)$
                // NP_w informs CB_u of new Sr_i
    endif
end

By the completion of function F0, a new service Sr_i of service provider SP_i, is now registered to the MEC network, with a unique identity number to be authenticated and distinguished among other service providers. Service provider's unique identity is issued and saved by the CSM. On the other hand, a network provider should have a list of all available registered service providers and must exchange it with other network providers, if any exists, within the same region. A network provider also must continuously update cloud brokers with the newly registered or deactivated service providers. Cloud brokers by return, will be aware and updated with service providers' status in their own area, which enhances service provider selection process by users. This minimizes the communication overhead between the participating entities in the MEC network.

2.6.2. Job request scenario algorithm

Algorithm 2, shows the job request scenario algorithm steps. A service user is expected to request one of the previously mentioned job types from a cloud broker, which in return searches for an appropriate service provider. If SU_j did not send rated SLA for its previously ended job in the MEC network, this user is penalized by being prohibited from requesting further jobs through function E1, described in section 2.6.4, until it sends the required rated SLA. Otherwise, CB_u asks SU_j to choose from a priority list its preferences, as shown in algorithm 2. Upon SU_j feedback, CB_u sends a recommendation list of available service providers, with respect to the chosen priorities [32]. SU_j chooses a service provider and informs the CB_u of its choice.

Consequently, CB_u checks the chosen service provider processing capacity $PC_i(Sr_i)$ limit. If $PC_i(Sr_i)$ is not maximum, the cloud broker starts direct communication between the service provider and user. Given that, each job will have a separate SLA, chosen Sr_i sends an initial SLA, SLA_{ia}_I, to SU_j for approval. Upon SLA approval by both parties, requested job processing starts, were Sr_i informs SU_j of the job start time; Pa_{st}. Job processing is ended or terminated in one of the previously mentioned four states, (P_{E1}, P_{E2}, P_{T1}, P_{T2}) by either the Sr_i or SU_j. Consequent steps take place accordingly as stated in Figure 2. In all cases, SU_j should send rated SLA_{ia}_R.

While TR_{ia} (time compliance, equation 2), is computed by Sr_i, it should be approved by SU_j within a time threshold, to avoid holding an opened transaction for a long time intentionally by any entity. In case, SU_j requests TR_{ia} correction, Sr_i should reply with the corrected TR_{ia}within a time threshold, otherwise complain function F7 (detailed in section 2.6.3) is called. Upon TR_{ia}, agreement, function F6 (algorithm 4) is called to check the validity of rated SLA_{ia}_R (depicted in section 2.6.3). The rated SLA_{ia}_R is encrypted by the public key of the CSM, which will handle trust computation process. This is performed using asymmetric encryption techniques, such as public key infrastructure (PKI) [33]. Using PKI, ensures secure trust computation, information integrity and confidentiality, while securing future user interactions [34].

Algorithm 2: Job request processing scenario-function F1.

```
Algorithm code: F1()
Description: Job Request and Execution Protocol
1.   Input: job_request by SU_j.
2.   Output: job_request of SU_j executed.
3.   SU_j→CB_u:job_request = job_type: (storage | processing | both)
4.   CB_u: checks
5.        if  E1_SU_j = true then
6.            execute Penalty function E1 // lazy SU_j
7.            exit ()
8.         endif
9.   CB_u→SU_j: priority_list of job_type
10.      where priority_list = (P(Sr_i) | storage capacity)
11.  SU_j→CB_u: returned priority_list answer
12.  CB_u→SU_j: recommendation_list of Sr_i with respect to priority_list answer
13.  SU_j→CB_u: chosen Sr_i
14.  CB_u: checks
15.          if  PC(Sr_i) = maximum then
16.              chosen Sr_i rejected
17.              return to step 12, excluding chosen Sr_i
18.           else
19.              CB_u→Sr_i: job_request of SU_j
20.          endif
21.  Sr_i :
22.          if Sr_i refuses job_request then
23.             Sr_i→CB_u: job_request rejected
```

```
24.             return to step 12, excluding chosen $Sr_i$
25.       else
26.          $Sr_i \rightarrow SU_j$: $SLA_{ia}\_I$ for approval
27.              where $SLA_{ia}\_I$ = ($SE_{ia}$, $SC_{ia}$, $SM_{ia}$, $SS_{ia}$)  // initial SLA
28.          $SU_j \rightarrow CB_u$: $SLA_{ia}\_I$
29.        endif
30. $CB_u$:
31.      if $SLA_{ia}\_I$ = approved then
32.            $CB_u \rightarrow Sr_i$: approved $SLA_{ia}\_I$
33.      else
34.            return to step 12, excluding chosen $Sr_i$
35.      endif
36. $Sr_i$: assigns
37.         job_type of $SLA_{ia}\_I$ to "a"
38.         issues $Pa_{st}$
39.                  where $Pa_{st}$=(start_date_time of process "a")
40. $Sr_i \rightarrow SU_j$: $Pa_{st}$ // informs $SU_j$ of start of processing
Case 1: Service provider ends job_request
41. $Sr_i \rightarrow SU_j$: $P_E$
42.      where $P_E = P_{E1} | P_{E2}$= ($Pa_{et}$, $TR_{ia}$, $T_v$, $Pa_{et}$, $Pa_{st}$)
43.                & $Pa_{et}$= (end_date & time of process $a_o$)
44.   $SU_j$: checks      // within time_threshold
45.       if   $T_v$ = approved then
46.            $SU_j \rightarrow CB_u$: ($E_n$ ($SLA_{ia}\_R$) $CSM_{Pu}$),
47.            where ($SLA_{ia}\_R$) = $rated$($SE_{ia}$, $SC_{ia}$, $SM_{ia}$, $SS_{ia}$)+ $TR_{ia}$_true+$P_{E1} | P_{E2}$
48.            execute Function F6
49.       else    // $TR_{ia}$_false
50.           $SU_j \rightarrow Sr_i$: requests $TR_{ia}$ correction,
51.             wait for corrected $P_E$,
52.           if received within time_threshold then
53.              return to step 44
54.          else
55.            execute Function F7 // complain function against $Sr_i$
56.          endif
57.       endif
Case 2: Service user terminates job_request
58. $SU_j \rightarrow Sr_i$: $P_T$
59.        where $P_T = P_{T1}$ | $P_{T2}$= (process termination request)
60.   $Sr_i$:
61.         computes $T_v$ & $TR_{ia}$
62.      $Sr_i \rightarrow SU_j$: $T_v$ & $TR_{ia}$
63.         $SU_j$: checks
64.             if  $T_v$ = approved then
65.                 $SU_j \rightarrow CB_u$: ($E_n$ ($SLA_{ia}\_R$) $CSM_{Pu}$),
66.                  where ($SLA_{ia}\_R$) = $rated$($SE_{ia}$, $SC_{ia}$, $SM_{ia}$, $SS_{ia}$)+ $TR_{ia}$_true +$P_{T1} | P_{T2}$
67.                  execute Function F6
68.             else            //  $TR_{ia}$_false
69.               $SU_j \rightarrow Sr_i$: requests $TR_{ia}$ correction,
70.                wait for corrected $P_T$,
71.              if received within time_threshold then
72.                 return to step 63
73.             else
74.                execute Function F7// complain function
75.            endif
76.          endif
77.           $CB_u$: executes Function F2
78.       endif
79. end
```

Upon the completion of function F1, CB_u gathers all rated SLAs of Sr_i, per computational interval and starts its computation process as described below.

2.6.3. Cloud broker computational algorithms

Functions (F2, F6 and F7) are executed by the cloud broker, as explained below. In Function F2 presented in algorithm 3, CB_u is responsible to collect all the rated SLA_{ia}_R per Sr_i, counts and sends them as a batch of rated SLA's (SLA_{ia}_Rn) to the CSM periodically for trust computation. This batch is sent encrypted by the public key of the CSM and stamped with the date of F2 function execution "Yd_{ia}".

Algorithm 3: Cloud broker computational-function F2

Algorithm code: F2.
Description: Cloud broker computational function.
Executed by the cloud broker.
Input: $E_n\,(SLA_{ia}\_R)\,CSM_{Pu}$ per Sr_i
Output: Batch of $(E_n\,(A, SLA_{ia}\_Rn)\,CSM_{Pu})$ + Yd_{ia} per Sr_i // Yd_{ia} date of computation
let $(E_n\,(SLA_{ia}\_R)\,CSM_{Pu})$ be referred to as SLA_{ia}_Rn
CB_u:
    A= *count* SLA_{ia}_Rn
    assign Yd_{ia} to current_date
$CB_u \rightarrow CSM$: Batch of $(E_n\,(A, SLA_{ia}\_Rn)\,CSM_{Pu})$ + Yd_{ia}
end

Algorithm 4 of Function F6, is called by function F1, where CB_u checks if SU_j had sent rated SLA of its last transaction. If not, then CB_u executes penalty function E1 to penalize SU_j, for its laziness. Function F7 presented in algorithm 5, is called by function F1, where a CB_u checks whether Sr_i had sent corrected TR_{ia} or not, within a time_threshold. If not sent, CB_u broadcasts message M_{TR} accordingly to all network providers, in order to stop dealing with the relevant Sr_i until sending corrected TR_{ia}. The network provider broadcasts M_{TR} to other cloud brokers as stated in algorithm 5.

Algorithm 4: Cloud broker checks SLA_R validity

Algorithm code: F6
Description: Cloud broker checks SLA_{ia}_R.
Executed by cloud broker.
Input: SLA_{ia}_R
Output: E1_SU_j = true or E1_SU_j = false
CB_u:
    if SLA_{ia}_R = null *then* // did not send SLA_{ia}_R
        generate E1_SU_j = true
        execute Penalty function E1
    else
        E1_SU_j = false
    endif
end

Algorithm 5: Cloud broker complain function against service provider

Algorithm code: F7
Description: Complain function against service provider.
Executed by cloud broker.
Input: TR_{ia} correction=null.
Output: Cloud broker stops sending new job_request of any
SU_j to Sr_i.
$CB_u \rightarrow Sr_i$: requests TR_{ia} correction
CB_u: stops sending new job_request to Sr_i until TR_{ia}
correction sent
$CB_u \rightarrow NP_w$: broadcast M_{TR}
    where M_{TR}=(pending TR_{ia} correction, no new
job_requests)
$NP_w \rightarrow$CB: broadcast M_{TR}
end

2.6.4. Penalties computational algorithms

Three penalty functions (E1, E2, E3) introduced in algorithms 6, 7 and 8 respectively, are described below. Algorithm 6 describes penalty function E1, which is called by function F6, in case SU_j did not send rated SLA_{ia}_R. In this function, a network provider broadcasts warning message M_{SLA} to all cloud brokers, to warn them not to accept any job requests from SU_j until sending the rated SLA_{ia}_R as shown in algorithm 6.

Algorithm 6: Penalty function E1

Algorithm code: E1.
Description: Service user denied sending rated SLA.
Executed by cloud broker.
Input: E1_SU_j=true.
Output: *requests* SLA_{ia}_R, job_request rejected
CB_u:
    job_request *rejected* of SU_j

5. $CB_u \rightarrow NP_w(SU_j)$: M_{SLA}
6. *where* M_{SLA}=(pending SLA_{ia}_R, job_request rejected)
7. $NP_w(SU_j) \rightarrow CB$: *broadcasts* M_{SLA} to nearby CB_u, ∀ u,
8. $CB \rightarrow SU_j$: E1_SU_j, *requests* SLA_{ia}_R
9. end

Algorithm 7 presents penalty function E2, were it's called by function F0, in case CSM discovers that a previously registered Sr_i, is trying to register as a new provider to the MEC network. Hence, CSM imposes penalty E2, which decreases Sr_i processing performance value, in the first attempt. In case this action is repeated again, malicious Sr_i is temporarily blocked from accessing the MEC network by penalty E5. Consequently, CSM sends warning message M_y to the involved CB_u and requests from it to find a replacement service provider, to delegate malicious Sr_i tasks to a newly selected provider. CB_u broadcasts this message to other cloud brokers and service providers, in order not to deal with malicious service provider temporarily.

Algorithm 7: Penalty function E2
Algorithm code: Penalty function E2.
Description: Service provider re-register attempt to MEC network.
Executed by CSM.
1. Input: E2_Sr_i
2. Output: P(Sr_i) decreased or system temporarily block for accused Sr_i.
3. CSM:
4. join_request *rejected*
5. *generate* E2_(Sr_i) // Penalty E2
6. *count* E2_Sr_i++
7. *if* E2_Sr_i =1 *then*
8. P(Sr_i)= P(Sr_i)-0.01
9. *execute* function F5
10. $CSM \rightarrow CB_u \rightarrow Sr_i$: ($E_n$(P($Sr_i$), T_n(Sr_i) $Sr_{i_{Pu}}$)
11. *elseif*
12. E2_Sr_i > 1 *then*
13. *generate* E5_Sr_{i1} // system temporarily block from the network of Sr_{i1} for X-days
14. $CSM \rightarrow CB_u \rightarrow Sr_{i1}$: E5_Sr_{i1}
15. $CSM \rightarrow CB_u(Sr_i)$: *requests* new Sr_{i2} selection|| M_y
16. *where* M_y=[compromised Sr_{i1}]
17. $CB_u(Sr_i) \rightarrow$ CB & Sr: *broadcasts* M_y to nearby CB_u & Sr_i, ∀ i & u
18. CB_u: *selects* Sr_{i2}
19. $CB_u \rightarrow Sr_{i2}$: job_delegation_request || M_y
20. *where* job_delegation_request=(job_request)
21. *endif*
22. end

On the other hand, penalty function E3, shown in algorithm 8, is called by function F4, in two cases; case1: a CB_u postpones sending the total number SLA_{ia}_R, "A", per Sr_i; case2: CB_u adjusts its computation time (Yd_{ia}) as a malicious action, during its processing of function F2. In both cases, the CB_u is claimed to be accused. CB_u is warned by warning number "W1", and a message is sent to it by the CSM. If one of these actions is repeated again, CSM sends warning number "W2" to the accused CB_u. If the CB_u performs any of these malicious actions for the third time, then penalty E4 is imposed by CSM, which deactivates CB_u from accessing the MEC network for a predefined time (X-days).

Algorithm 8: Penalty function E3
Algorithm code: E3.
Description: Monitoring cloud broker actions.
Executed by CSM.
1. Input: E3_CB_u.
2. Output: W1_CB_u or W2_CB_u or system temporarily block for CB_u.
3. CSM:
4. *generate* E3_CB_u
5. *count* E3_CB_u++
6. *if* E3_CB_u =1 *then*
7. *generate* W1_CB_u
8. $CSM \rightarrow CB_u$: W1_CB_u
9. *elseif*
10. E3_CB_u =2
11. *generate* W2_CB_u

```
                 CSM→CBu: W2_CBu
             else
                generate E4_CBu1
                CSM→CBu: E4_CBu1
                CSM: performs system temporarily block from the network of CBu1 for X-
                days
                requests all stored (En (A,SLAia_R) CSMPu) from CBu1
                CSM→NPw(CBu): requests new CBu2 selection || Mx
                      where Mx=[compromised CBu1]
                      NPw(CBu): selects nearby CBu2
                      NPw(CBu)→ CBu2: Mx
                      NPw(CBu)→CSM: newly selected CBu2
                      CBu2→ Sri & SUj:broadcasts Mx
           endif
end
```

In case of CB_u deactivation, CSM requests from the involved network provider to find a subsequent cloud broker and delegates all its functions to the newly selected cloud broker, which continues all the pending jobs. It also broadcasts a warning message (M_x) to all surrounding service providers and users, to inform them of the compromised CB_u. As shown, the penalty functions update the participating entities in case a malicious participant is discovered, where this entity is banned from accessing the MEC network. This optimizes the security of interactions on the MEC network.

2.6.5. Trust computational algorithms

Figure 4 shows the processing performance computation steps of Sr_i. Upon completion of function F2 by the cloud broker, the processing performance evaluation of Sr_i, function F4 presented in algorithm 9, is executed by the CSM per computational interval, given that "A > 0" (Sr_i had received jobs). In case a service provider is working with more than one cloud broker, CSM could authenticate each provider by its unique identity. Function F4 calls two other functions; function F3 (algorithm 10) for processing throughput computation, and function F5 (algorithm 11), that evaluates service provider trust status.

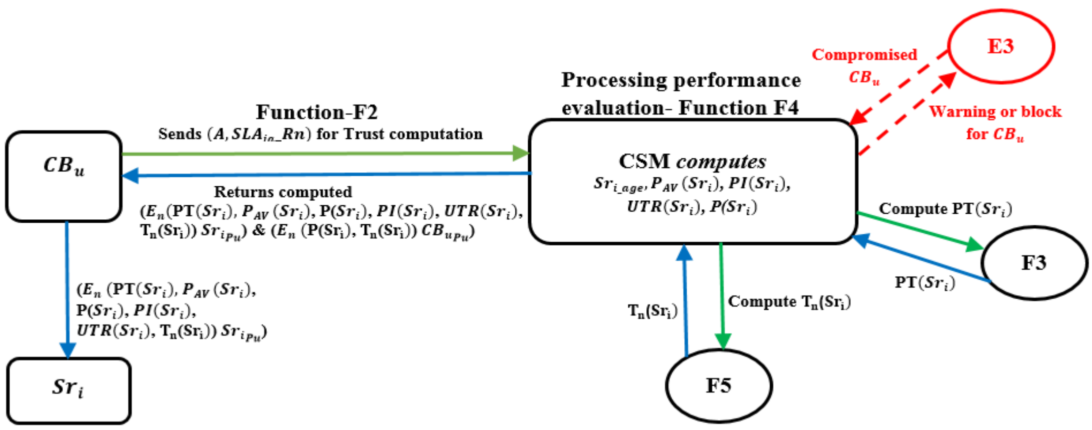

Figure 4. Processing performance computation function-F4

Trust evaluation and all its parameters are evaluated by the CSM for three main reasons: 1- to ensure accurate and fair trust computation for service providers, 2- to guarantee service provider data security and confidentiality, 3- it helps service requesters to reveal trust values remotely prior starting their interactions. Sr_i age is computed to show its processing lifetime in the MEC network. Trust computation begins by computing P_{E1}_T, P_{E2}_T, P_{T1}_T, P_{T2}_T. Each of these parameters indicates the final status per process "a" received by Sr_i. While the computed time difference $TR_{ia} \geq 0$, the average processing success rate P_{AV} (Sr_i) is computed. This is because there could be successful processes, in spite that Sr_i had exceeded the estimated agreed time SE_{ia}, but the service user had been patient enough. Therefore, this case is not considered as a fully successful job and could indicate that the predetermined processing capacity is not accurate for this Sr_i. However, this raises the processing throughput value for Sr_i computed by function F3, algorithm 10.

While the processing incompliance $PI(Sr_i)$ and user termination ratio $UTR(Sr_i)$ are computed, in case $UTR(Sr_i)$ exceeds user termination threshold, a warning is generated and sent to the relevant Sr_i via the CB_u. This is to alarm Sr_i of its high user termination ratio. Consequently, the processing performance P(Sr_i) is computed, then function F5 is executed, to get Sr_i trust status. These results are encrypted by CSM and sent to Sr_i via its CB_u. Processing throughput computation, function F3 (algorithm 10), is computed based on the number of successful processes P_{E1}_T executed by Sr_i, (equation (6)). Based on the computed processing performance P(Sr_i), Sr_i is assigned one of six trust states by function F5, as shown in algorithm 11.

Algorithm 9: Processing performance computation-function F4

Algorithm code: F4.
Description: Processing performance computation function.
Executed by CSM.

Input: $(E_n\,(A, SLA_{ia}\_Rn)\,CSM_{Pu}) + Yd_{ia}$
Output: P(Sr_i) computed
$CB_u \rightarrow CSM$: batch of $(E_n\,(A, SLA_{ia}\_Rn)\,CSM_{Pu}) + Yd_{ia}$
CSM:
  decrypt $(E_n\,(A, SLA_{ia}\_Rn)\,CSM_{Pu})$
    if Yd_{ia} exceeded last_day of month *then*
      execute Penalty function E3
    endif
      if $Yd_{ia} \geq Pa_{et}\_last$ *then*
        for a=1 to A
          compute
            P_{E1}_T, P_{E2}_T, P_{T1}_T, P_{T2}_T,
            Sr_{i_age}= Current_date - Sr_{i_birth}
              if $P_{E1}\_T \geq 1$ *then*
                execute Function F3 // compute PT(Sr_i)
                  if $TR_{ia} \geq 0$ *then*
                    compute $P_{AV}(Sr_i) = \frac{P_{E1}\_T}{A}$ // equation (3)
                  endif
              elseif $P_{E2}\_T \geq 1$ or $P_{T1}\_T \geq 1$
                compute $PI(Sr_i) = \frac{P_{E2}\_T + P_{T1}\_T}{A}$ // equation (4)
              elseif $P_{T2}\_T \geq 1$
                compute $UTR(Sr_i) = \frac{P_{T2}\_T}{A}$ // equation (5)
                if $UTR(Sr_i) \geq$ user termination threshold *then*
                  generate W1_$\mathrm{Sr_i}$
                  CSM→ CB_u: W1_$\mathrm{Sr_i}$
                  $CB_u \rightarrow \mathrm{Sr_i}$: W1_$\mathrm{Sr_i}$
                endif
            else
                PT(Sr_i)=0, $P_{AV}(Sr_i)$=0, $PI(Sr_i)$=0, $UTR(Sr_i)$=0
            endif
        endfor
      else
        CB_u=compromised
        execute Penalty function E3
      endif
compute P($\mathrm{Sr_i}$) // equation (7)
execute function F5
CSM→CB_u: $(E_n\,(\mathrm{PT}(Sr_i), P_{AV}(Sr_i), \mathrm{P}(Sr_i), PI(Sr_i), UTR(Sr_i), \mathrm{T_n}(\mathrm{Sr_i}))\,Sr_{i_{Pu}})$ &
  $(E_n\,(\mathrm{P}(\mathrm{Sr_i}), \mathrm{T_n}(\mathrm{Sr_i}))\,CB_{u_{Pu}})$
$CB_u \rightarrow Sr_i$: $(E_n\,(\mathrm{PT}(Sr_i), P_{AV}(Sr_i), \mathrm{P}(Sr_i), PI(Sr_i), UTR(Sr_i), \mathrm{T_n}(\mathrm{Sr_i}))\,Sr_{i_{Pu}})$
end

Algorithm 10: Processing throughput computation-function F3

Algorithm code: F3.
Description: Processing throughput computation function.
Executed by CSM.

Input: $\mathrm{P_{E1}}\_T$.
Output: PT($\mathrm{Sr_i}$) // processing throughput computation per computational interval for $\mathrm{Sr_i}$.
CSM :
  if ($\mathrm{P_{E1}}\_T \leq 5{,}000$) *then*
    Points(Sr_i) = Ceil ($\mathrm{P_{E1}}\_T/10{,}000$)
    elseif ($\mathrm{P_{E1}}\_T \leq 10{,}000$) *then*
      Points(Sr_i) = 0.6
      elseif ($\mathrm{P_{E1}}\_T \leq 100{,}000$) *then*
        Points(Sr_i) = 0.7

```
            elseif  (PE1_T ≤ 1,000,000) then
               Points(Sri)  = 0.8
               else if  (PE1_T ≤ 10,000,000) then
                  Points(Sri)  = 0.9
                  else
                     Points(Sri)  = 1
                     return PT(Sri)= Points(Sri)
      endif
end
```

Algorithm 11: Trust status computation algorithm-function F5

```
Algorithm code: F5.
Description: Trust status computation function.
Executed by CSM.
Input: P(Sri)
Output: Tn(Sri)
CSM:
         If P(Sri) ≤0 then
                 Tn(Sri)= "untrusted service provider"
             elseif P(Sri) ≤ 0.3 then
                     Tn(Sri)= "weak service provider"
             elseif P(Sri) ≤ 0.5 then
                     Tn(Sri)= "average service provider"
             elseif P(Sri) ≤ 0.7 then
                     Tn(Sri)= "good service provider"
             elseif P(Sri) ≤ 0.9 then
                     Tn(Sri)= "very good service provider"
             else
                     Tn(Sri)= " excellent service provider"
         endif
return Tn(Sri)
end
```

Upon the completion of the above functions, each service provider will be assigned a trust status based on its computed processing performance value. This trust status disseminates service provider's processing performance during its service provisioning in the MEC network.

3. RESULTS AND DISCUSSION

Simulation results of the proposed architecture are shown in section 3.1, while the efficiency and effectiveness of proposed scheme are discussed in section 3.2. Section 3.3 presents a comparison between the proposed architecture and some previous protocols.

3.1. Simulation results

Simulation of the proposed model was performed using MATLAB program. Simulation setup:

- five different service providers were considered, Sr_i = {1,2,3,4,5}.
- initial trust value for each Sr_i= zero.
- all Sr_i received the same number of job requests, in one job_type{Job3}, from various service users.
- computation intervals = 5 (1 month duration each)
- P_{E1}_T, P_{E2}_T, P_{T1}_T, P_{T2}_T values: were assigned to each Sr_i according to uniform random number generation per month.
- hardware PC configuration = core i7, RAM 6 GB and hard disk 1 Tera.

Trust evaluation was performed over the five months. Sr_i is evaluated according to its processing performance in its predefined job type. The processing throughput PT(Sr_i) results for each service provider per "m" month are shown in Figure 5. While the average processing success rate P_{AV} (Sr_i) results are given Figure 6.

Upon measuring the processing throughput and average success rate, the processing performance P(Sr_i) is computed as presented in Figure 7 and the relevant trust status per Sr_i over the five months, is shown in Table 3. Figure 8 reveals the processing incompliance (failure ratio) PI(Sr_i) and Figure 9 shows the user termination ratio per Sr_i. Results show that service providers 1 and 2, trust status had improved gradually by time, because of their improvement in terms of processing throughput and average processing success rate over the five months. With this improve, processing incompliance and user termination ratio, had decreased as shown Figures 8 and 9. Service provider 3 trust status kept varying by time, within a good to average range. While service provider 4 maintained its good trust status over the five months, however its processing incompliance acts as a major drawback as illustrated in Figure 8. Service provider 5, kept its excellent processing performance over the five months since its processing incompliance and user termination ratio are very low. Thus, simulation results could identify the processing performance of all five service providers, together with their processing incompliance and user termination ratios, showing their respective trust status.

3.2. Efficiency and effectiveness of the proposed architecture

Analysis of the proposed architecture shows that the evaluation time is considerably low, due to the simplicity of the used equations. Processing performance evaluation and trust status results are updated periodically by CSM (fully trusted entity), which increases results credibility. In addition, maintaining a history record decreases service provider trust evaluation time, since it's performed in an accumulative manner. A service provider is registered only once to the MEC network using its unique credentials. This encounters attacks such as fake or malicious service providers, who could deceive users by hiding their bad history.

As the number of service providers increases, the proposed architecture could still distinguish each service provider using its assigned trust status and processing performance value. This validates service providers' computational services trust level and history in the MEC network, which promotes for trusted and secured transactions. A service user is also given a recommendation list of available service providers to choose from, according to user's computational requirements and preferences.

3.3. Comparison with previous protocols

Table 4 shows a comparison of the proposed architecture evaluation parameters with previous works. The major limitations/discussion for each one of them.

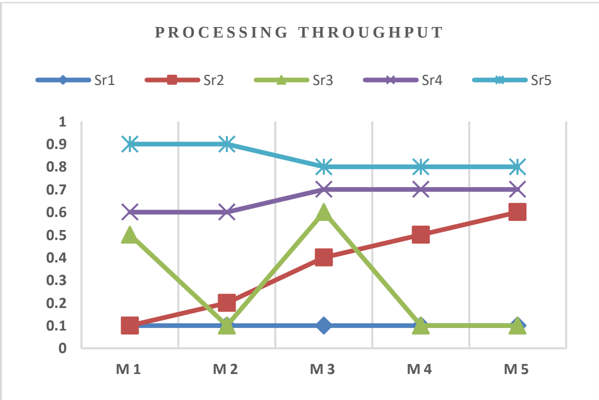

Figure 5. Processing throughput per service provider

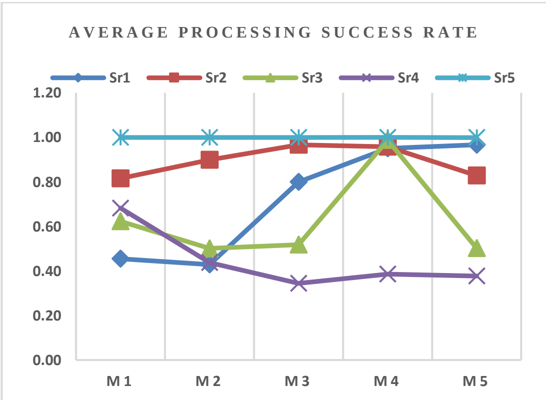

Figure 6. Average processing success rate per service provider

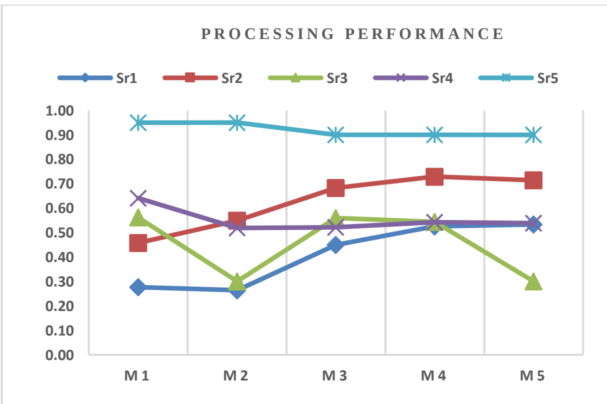

Figure 7. Processing performance per service provider

Table 3. Trust status per service provider in "m" months

Sr_i	M1	M2	M3	M4	M5
Sr_1	Weak	Weak	Average	Good	Good
Sr_2	Average	Good	Good	Very good	Very good
Sr_3	Good	Average	Good	Good	Average
Sr_4	Good	Good	Good	Good	Good
Sr_5	Excellent	Excellent	Very good	Very good	Very good

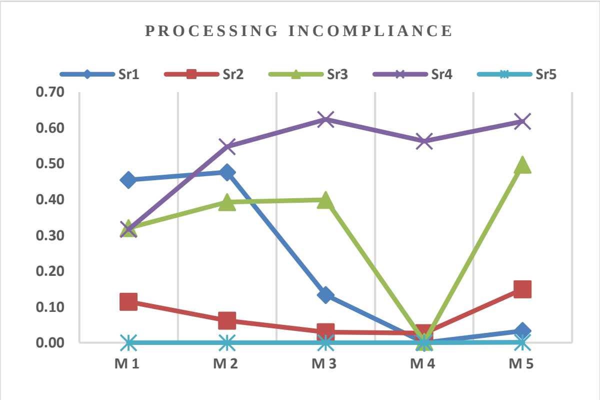

Figure 8. Processing incompliance per service provider

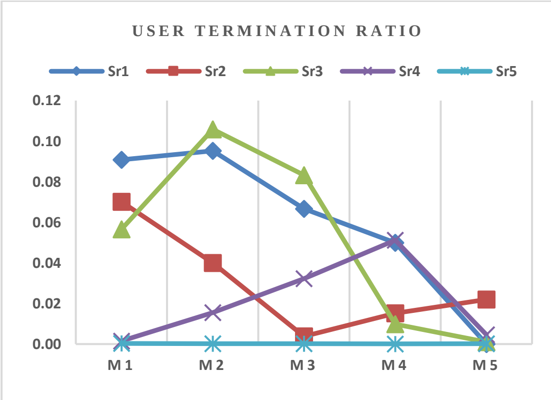

Figure 9. Service providers' user termination ratio

Table 4. Comparison of the proposed architecture and previous work

Protocol	Service Provider Assessed Parameters	Evaluation Domain	Trust Update Static	Trust Update Dynamic	Limitations/Discussion
[17] 2018	Security methods applied in terms of: i) authentication, ii) firewall systems; iii) encryption mechanisms, iv) intrusion detection	Computation-based	√		Did not consider sudden attacks that could occur to a node, such as hacking.
[18] 2020	Performance in terms of: -identity authentication -hardware capabilities -interactions' behavior	Reputation-based (Direct/indirect trust)		√	Depended only on users' opinions. Collusion attack may occur.
[19] 2018	Performance in terms of: -identity authentication -capabilities (availability, response time, throughput, deployed hardware) -interactions' behavior	Feedback-based & computation-based		Partial	Trust computation is performed by an unknown entity, which makes trust results sharing difficult. Trust computation is complicated and time consuming.
[20] 2018	Analyze traffic flows between two communicating entities.	Computation-based		√	No processing parameters are evaluated.
Proposed Protocol 2020	Processing performance in terms of: -processing throughput -average processing success rate -processing incompliance -user termination ratio.	Computation-based		√	No human interaction involved, which guarantees results credibility. History capturing decreases computational overhead and limits re-register attack. Dynamic updating of results shows recent trust status of a service provider in the MEC network. Proposed a penalty system to track malicious entities.

4. CONCLUSION AND FUTURE WORK

Trust was evaluated by computing the processing performance of a service provider, through gaining its average processing success rate and processing throughput. However, processing incompliance and user termination ratio were computed, to accurately determine service providers' performance in the MEC network. The proposed penalty system provided a close monitoring to the participating entities in the MEC network. By capturing the historical trust results, there is no need to evaluate a service provider trust status before the start of each interaction. Thus, gaining accurate and fair trust results with less computation overhead and minimal human interference.

Simulation results showed that, the higher average processing success rate and throughput, the better processing performance and trust status evaluation gained for a service provider. On the other hand, results illustrated that high processing incompliance or user termination ratio, are reflected in a low processing performance value for a service provider. Thus, maintaining service users' reliability and securing future interactions in the MEC environment. For future work, we plan to evaluate service providers' deployed hardware and software resources. In this context, security measures, scheduling algorithms, fault tolerant protocols deployed by a service provider should be considered during trust evaluation. On the other hand, the number and types of warnings imposed on a service provider, due to performing unauthorized actions should also be considered and analyzed in the future. Given that all acting entities are registered in the network through the network provider, a network provider could act as a cloud broker or even a service provider. On the other hand, a cloud broker could also act as a service provider. However, it will be recommended to review the trust evaluation parameters for these acting entities.

REFERENCES

[1] S. N. Shirazi, A. Gouglidis, A. Farshad, and D. Hutchison, "The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2586–2595, Nov. 2017, doi: 10.1109/JSAC.2017.2760478.

[2] H. Li, G. Shou, Y. Hu, and Z. Guo, "Mobile edge computing: progress and challenges," in *Proceedings - 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2016*, 2016, pp. 83–84, doi: 10.1109/MobileCloud.2016.16.

[3] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: fog computing, cloudlet and mobile edge computing," in *GIoTS 2017 - Global Internet of Things Summit, Proceedings*, 2017, doi: 10.1109/GIOTS.2017.8016213.

[4] B. Liang, "Mobile edge computing," in *Key Technologies for 5G Wireless Systems*, V. W. S. Wong, R. Schober, D. W. K. Ng, and L.-C. Wang, Eds. Cambridge University Press, 2017, pp. 76–91.

[5] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: a Survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb. 2018, doi: 10.1109/JIOT.2017.2750180.

[6] H. El-Sayed *et al.*, "Edge of things: the big picture on the integration of edge, IoT and the cloud in a distributed computing environment," *IEEE Access*, vol. 6, pp. 1706–1717, 2017, doi: 10.1109/ACCESS.2017.2780087.

[7] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017, doi: 10.1109/ACCESS.2017.2685434.

[8] J. Tan, R. Gandhi, and P. Narasimhan, "Challenges in security and privacy for mobile edge-clouds," Parallel Data Laboratory Carnegie Mellon University Pittsburgh, 2014.

[9] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: the communication perspective," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017, doi: 10.1109/COMST.2017.2745201.

[10] P. Mach and Z. Becvar, "Mobile edge computing: a survey on architecture and computation offloading," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017, doi: 10.1109/COMST.2017.2682318.

[11] M. B. Monir, M. H. Abdelaziz, A. A. Abdelhamid, and E. S. M. Ei-Horbaty, "Trust management in cloud computing: a survey," in *2015 IEEE 7th International Conference on Intelligent Computing and Information Systems, ICICIS 2015*, 2016, pp. 231–242, doi: 10.1109/IntelCIS.2015.7397227.

[12] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *Proceedings of the 10th International Conference on Intelligent Systems and Control, ISCO 2016*, 2016, doi: 10.1109/ISCO.2016.7727082.

[13] N. Makitalo, A. Ometov, J. Kannisto, S. Andreev, Y. Koucheryavy, and T. Mikkonen, "Safe and secure execution at the network edge: a framework for coordinating cloud, fog, and edge," *IEEE Software*, pp. 1–1, 2018, doi: 10.1109/MS.2018.110164708.

[14] J. Yuan and X. Li, "A reliable and lightweight trust computing mechanism for IoT edge devices based on multi-source feedback information fusion," *IEEE Access*, vol. 6, pp. 23626–23638, 2018, doi: 10.1109/ACCESS.2018.2831898.

[15] V. Vassilakis, E. Panaousis, and H. Mouratidis, "Security challenges of small cell as a service in virtualized mobile edge computing environments," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9895 LNCS, Springer International Publishing, 2016, pp. 70–84.

[16] F. Pop, C. Dobre, B. C. Mocanu, O. M. Citoteanu, and F. Xhafa, "Trust models for efficient communication in mobile cloud computing and their applications to e-commerce," *Enterprise Information Systems*, vol. 10, no. 9, pp. 982–1000, Dec. 2016, doi: 10.1080/17517575.2015.1100756.

[17] F. J. Mora-Gimeno, H. Mora-Mora, D. Marcos-Jorquera, and B. Volckaert, "A secure multi-tier mobile edge computing model for data processing offloading based on degree of trust," *Sensors (Switzerland)*, vol. 18, no. 10, p. 3211, Sep. 2018, doi: 10.3390/s18103211.

[18] X. Deng, J. Liu, L. Wang, and Z. Zhao, "A trust evaluation system based on reputation data in Mobile edge computing network," *Peer-to-Peer Networking and Applications*, vol. 13, no. 5, pp. 1744–1755, Feb. 2020, doi: 10.1007/s12083-020-00889-3.

[19] X. Ma and X. Li, "Trust evaluation model in edge computing based on integrated trust," in *ACM International Conference Proceeding Series*, 2018, doi: 10.1145/3302425.3302491.

[20] Y. Ruan, A. Durresi, and S. Uslu, "Trust assessment for internet of things in multi-access edge computing," in *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 2018, vol. 2018-May, pp. 1155–1161, doi: 10.1109/AINA.2018.00165.

[21] M. H. Ur Rehman, P. P. Jayaraman, S. Ur Rehman Malik, A. Ur Rehman Khan, and M. M. Gaber, "RedEdge: a novel architecture for big data processing in mobile edge computing environments," *Journal of Sensor and Actuator Networks*, vol. 6, no. 3, p. 17, Aug. 2017, doi: 10.3390/jsan6030017.

[22] S. M. Habib, S. Hauke, S. Ries, and M. Mühlhäuser, "Trust as a facilitator in cloud computing: a survey," *Journal of Cloud Computing*, vol. 1, no. 1, pp. 1–18, 2012, doi: 10.1186/2192-113X-1-19.

[23] H. Bangui, S. Rakrak, S. Raghay, and B. Buhnova, "Moving to the edge-cloud-of-things: recent advances and future research directions," *Electronics (Switzerland)*, vol. 7, no. 11, p. 309, Nov. 2018, doi: 10.3390/electronics7110309.

[24] J. Huang and D. M. Nicol, "Trust mechanisms for cloud computing," *Journal of Cloud Computing*, vol. 2, no. 1, p. 9, 2013, doi: 10.1186/2192-113X-2-9.

[25] I. Oteyo, D. P. Mirembe, and M. P. Nampala, "Scaling trust and reputation management in cloud services," *International Journal of Applied Science and Technology*, vol. 6, pp. 50–57, 2016.
[26] C. Uikey and D. S. Bhilare, "A broker based trust model for cloud computing environment," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 11, 2013.
[27] R. Shaikh and M. Sasikumar, "Trust model for measuring security strength of cloud computing service," *Procedia Computer Science*, vol. 45, no. C, pp. 380–389, 2015, doi: 10.1016/j.procs.2015.03.165.
[28] V. Vassilakis *et al.*, "Security analysis of mobile edge computing in virtualized small cell networks," in *IFIP Advances in Information and Communication Technology*, vol. 475, Springer International Publishing, 2016, pp. 653–665.
[29] A. Gholami and M. G. Arani, "A trust model based on quality of service in cloud computing environment," *International Journal of Database Theory and Application*, vol. 8, no. 5, pp. 161–170, Oct. 2015, doi: 10.14257/ijdta.2015.8.5.13.
[30] A. Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: a complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289–330, Sep. 2019, doi: 10.1016/j.sysarc.2019.02.009.
[31] M. B. Monir, T. Abdelkader, and E. S. M. Ei-Horbaty, "Trust evaluation of service level agreement for service providers in mobile edge computing," in *Proceedings - 2019 IEEE 9th International Conference on Intelligent Computing and Information Systems, ICICIS 2019*, 2019, pp. 362–369, doi: 10.1109/ICICIS46948.2019.9014854.
[32] F. Zohra Filali and B. Yagoubi, "Global trust: a trust model for cloud service selection," *International Journal of Computer Network and Information Security*, vol. 7, no. 5, pp. 41–50, Apr. 2015, doi: 10.5815/ijcnis.2015.05.06.
[33] F. Meixner and R. Buettner, "Trust as an integral part for success of cloud computing," in *ICIW 2012*, 2012.
[34] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: a survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, Aug. 2019, doi: 10.1016/j.future.2019.02.050.