INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING
DOI: 10.46300/9106.2020.14.40 Volume 14, 2020

Cloud Based Mission Critical Calls at the Edge

I. Atanasov, E. Pencheva, A. Nametkov
Technical University of Sofia
Bulgaria

Received: March 1, 2020. Revised: July 1, 2020. Accepted: July 7, 2020. Published: July 9, 2020.

Abstract— Multi-access Edge Computing (MEC) technology outsources the cloud services at the edge of the mobile network for delay-sensitive, bandwidth hungry applications. The technology addresses the requirements of mission critical communications for ultralow latency and high reliability in a sustainable and affordable way. The paper studies MEC capabilities to handle mission critical calls exposing the network functions for traffic gating and rerouting. Following the RESTful approach to define MEC services, information flows, interfaces with data model and data format are presented. The injected latency by the service is theoretically evaluated.

Keywords—5G, Multi-access Edge Computing, Application Programming Interfaces, state models, bi-simulation, latency.

I. Introduction

MISSION critical communications are required when human life or any kind of critical infrastructure are at risk, and timely and reliable reaction is of great importance to avoid or mitigate the damages [1], [2]. Voice mission critical services offer functionality for engagement in a call of two or more users, group management, private calls between pairs of users, floor control, etc. The recent fifth generation (5G) mobile networks address the requirements of mission critical communications by offering high availability and flexibility, prioritization of users related to public safety, prioritization of traffic classes, and management of extreme conditions when the network is congested [3], [4]. In [5], the authors present and evaluate 5G architecture that enables mission critical applications. The vision and challenges of mission critical data communications and the 5G enabling technologies is provided in [6]. Possible strategies to achieve reliable and delay optimized mission critical networks are discussed in [7]. Approaches for mission critical communications aimed at search and rescue operations which are based on unmanned aerial vehicles are proposed in [8], [9]. In [10], the authors argue that shared access and network slicing enable support of mission critical applications. In [11], the authors propose an approach to provide mission-critical messaging services using MEC technology.

Mission Critical Push To Talk (MCPTT) service provides half duplex communication service with enhanced features suitable for mission critical scenarios. It enables establishment, maintenance and termination of communication path(s) among users, call management, monitoring on activity of separate sub-calls, and a mechanism to arbitrate between multiple simultaneous requests. The service is targeted at transport companies, fire brigades, police, ambulance as well as industrial and nuclear plants.

In this paper, we propose an approach to deploy mission critical voice applications at the 5G network edge. The approach enables programmability of mission critical voice services and does not require deployment of IMS (Internet Protocol Multimedia Subsystem) platform. The approach is based on REpresentational State Transfer (REST) architectural style, and defines the related resources, data model and supported interfaces. The feasibility study illustrates the approach practicality. The service processing time is assessed theoretically.

II. Mission Critical Voice Communications at the Edge

The MCPTT service enables users to gain access to the permission to talk by arbitrated manner and supports private calls between users in a group. Requirements of MCPTT service and service architecture are described in 3GPP TS 23.379, while the signaling procedures and protocols are defined in 3GPP TS 24.379. Both specifications require deployment of MCPTT service over IMS which is optimized to provide all types of multimedia services using internet protocols. The IMS functionality as a part of the core network ensures proper establishment and management of bearers for emergency sessions.

MCPTT service can be based on distributed cloud architecture at the network edge. The Multi-access Edge Computing (MEC) which is a key component of 5G networks addresses the requirements for resiliency and low latency and can further improve the MCPTT service performance. By outsourcing cloud capabilities at the edge of the mobile network, MEC enables session continuity, distributed transaction management and offloading the mobile device workload. MEC-based application traffic does not need to

The research is funded by Bulgarian Science Fund, grant KP-06-H37/18

travel between mobile devices and the cloud to be processed, which reduces the round-trip time. A hierarchical distributed MCPTT architecture which is based on IMS and MEC is suggested in [12].

The Key Performance Indicators (KPIs) of MCPTT service are defined in 3GPP TS 22.179. End-to-end access time is one of the most important criteria which is defined as the time between the user request to speak and the user gets a signal to start speaking. It includes IMS signaling procedures which introduce significant overhead. An analysis on impact of 5G network on MCPTT service KPIs is provided in [13].

In this paper, we propose MEC-based call control service which does not require IMS deployment. Our proposal eliminates the necessity of IMS signaling and uses the open access to distributed core functionality. Due to reduced signaling for session establishment the access time is significantly reduced. The proposed approach to mission critical call control is beneficial in cases of backhaul connection loss as the call related signaling is isolated at the edge.

Next sections describe the proposed call control functionality by typical use case, data types and information exchanged, the resource methods, state models and their formal verification.

III. Description of the Proposed Functionality

The proposed functionality for mission critical voice services can be provided as a mobile edge service by co-location of the MEC platform and distributed core network. The MEC deployment with distributed core functionality enables superior quality of experience and performance for mission critical scenarios [14]. With the service-based architecture of the core network, network functions are provided as services and can be virtualized. This architecture is aligned with the principles of MEC Application Programming Interfaces (APIs) framework. The MEC API framework provides the necessary functionality for mobile edge service registration, discovery, authentication, and authorization, which is also required for core network services. Mobile edge applications can be run on the same virtualized platform as the core network services which reduces the costs. The deployment of distributed core network components serves well mission critical communications as the communication with the operators centralized core is not necessary. The solution enables local management, offloading of the entire traffic, and provisioning of customized quality of service.

The mobile edge platform and applications can access core network functionality through Network Exposure Function (NEF) [15]. 5G NEF enables MEC functional entities to subscribe to relevant events and to be notified on event occurrence. It also allows mobile edge applications to provision prognostics about user mobility and activity to optimize network performance, as well as to handle quality of service based on policies.

We propose a new mobile edge service named Call Handling service which enables mobile edge applications to handle calls initiated by mobile subscribers. A mobile edge application notified about a call related event can determine how the call should be treated. It is possible for the mobile edge application to request call re-routing, call ending (termination), or call continue. The proposed mobile edge service is activated as a result of event in the network and the mobile edge application can indicate the call treatment prior the call establishment.

More specifically, the proposed mobile edge Call Handling service provides a mechanism for mobile edge applications to specify how calls should be handled for a specific number. Call handling instructions include the following:

- Call accepting – accepting calls from numbers which are not in a blacklist, i.e. completing the call to the original called party address.
- Call blocking – blocking calls from numbers that are in a blacklist, i.e. rejecting the call.
- Unconditional call forwarding – changing the called party address, i.e. the called number is changed to the forwarding number.
- Conditional call forwarding – changing the destination address of the call based on the state of the called party (busy, unreachable, no answer).
- Play audio - initiating audio with the calling party (e.g. playing an announcement), i.e. the call is handled by a voice system.
- Call transferring – transferring the voice call to another party, i.e. the call is transferred to another party during its active phase.

A typical mission critical scenario is a call to a certain service number. The service number is used to connect the calling subscriber to e.g. a Public Safety Agency. A dedicated mobile edge application being notified about the call may redirect the call to the appropriate personal based on e.g. location, time, availability etc. In alternative scenario, the mobile edge application may route the call to media server to play a message to the caller.

Fig.1 illustrates the flow for the call handling by a mobile edge application. As a pre-condition for call handling, the mobile edge application needs to have active subscription for call related events.

1. Using Call Handling API, the mobile edge application requests to subscribe for notifications about call related events.
2-3. The Call Handling service invokes the Nnef_EventExposure service of NEF, which provides support for event exposure, to create subscription.
4. The mobile edge application is notified about accepted subscription.
5. The user makes a call to the service number.
6-7. The NEF invokes the Nnef_EventExposure service to notify the Call Handling service about the event.
8-9. The application is also notified about the call event.

10. The application decides to redirect the call and requests the call redirection to specific address.
11-12. The Call Handling service invokes Nnef_Traffic-Influence service of NEF, which provides the ability to affect the traffic routing.
13. The call is redirected in the network.
14-15. The NEF invokes the Nnef_TrafficInfluence service to notify the Call Handling service about call redirection.
16. The application request for call redirection is acknowledged.

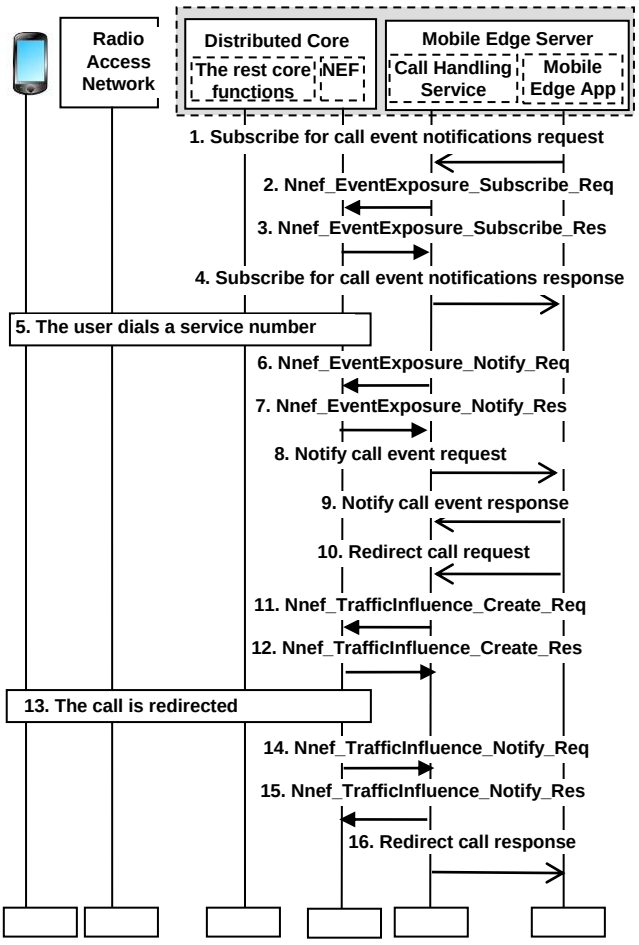

Fig.1 A mobile edge application having active subscription for call events, requests call redirection upon notification

IV. Data Model and API Definition

REST is architectural style for distributed applications. It takes a resource-based approach for interactions where resources represent entities (physical or logical). The resource identification is performed by its URI (Uniform Resource Identifier) which enables a global addressing scheme for resources and service discovery. Resources are decoupled from their representation and their content may be described in different format such as JSON, XML etc. each resource has a state which can be acted upon by four simple operations: create (HTTP POST), retrieve (HTTP GET), update (HTTP PUT) and delete (HTTP DELETE). Due to its properties like scalability, performance, and modularity REST is desirable in web service design [16].

Following the REST architectural style, all objects of the proposed Call Handling service are represented as resources organized in a tree structure. All resource URIs follow a common root that can be discovered using service discovery.

Fig.2 shows the resource structure of the proposed Call Handling mobile edge service.

The callHandlingRequests resource contains all requests for call handling. It supports GET method which retrieves a list of all request for call handling. The routingRequests resource is a container resource for all requests to (re-)route calls to address indicated by a mobile edge application. It supports POST method which creates a new resource representing a request for call (re-) routing and GET method which retrieves the list of all requests to (re-)route calls.

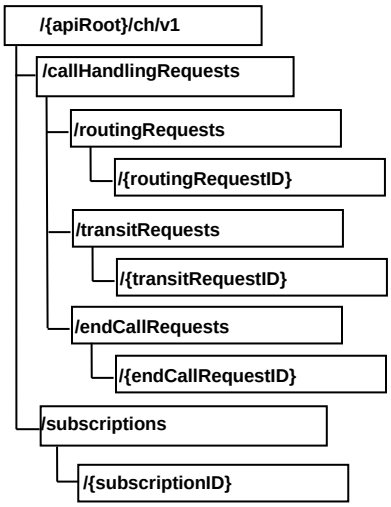

Fig.2 Resource structure of Call Handling service

When a mobile edge application wants to (re-)route the call to a specific address it sends a POST request to the routingRequests resource with message body containing the routingInfo data structure. The routingInfo data type describes the information about call routing and it is a structure of routingAddress (the address to which the call has to be (re-)routed), mediaInfo describing the media types to be used in each direction, the callingUser (the URI of the caller), and application instance ID and request ID. The Call Handling service answers with 201 Created response.

The routingRequestID resource represents an existing request for call (re-)routing. It supports GET method which retrieves information about the given request.

The transitRequests resource contains all requests to handle the call as transit e.g. to continue the call without any changes. It supports POST method which creates a new resource representing a request to continue with normal call handling and GET method which retrieves the list of all requests to handle the call as transit.

The transitRequestID resource represents an existing request for handling calls as transit. The supported method is GET which retrieves information about specified request.

The endCallRequets resource contains all requests to end the call which results in call termination in the network. The caller may receive an announcement. It supports POST method which creates a new resource representing a request to end the call and GET method which retrieves the list of all requests to end the call. When a mobile edge application decides to end the call, it sends a POST request to the endCallRequets resource with message body containing the endCallInfo data structure. The endCallInfo data type describes the information about the call ending and it is a structure of callingUser address, the URI indicating the location of the announcement to be played, the application ID and the request ID. The mobile edge Call handling service responds with 201 Created message.

The callEndRequestID resource represents an existing request to end the call and it supports GET method.

The subscriptions resource represents all subscriptions for notifications about call related events. It supports GET method which retrieves a list of all subscriptions and POST method which creates a new subscription. When a mobile edge application wants to create a new subscription, it sends a POST request to the subscriptions resource with message body containing eventSubscriptionInfo data structure. The eventSubscriptionInfo data type represents information about subscription to call related events. It is a structure of callback-Address, subscriptionID, filterCriteria and expiryDeadline, where: callbackAddress is URI selected by the mobile edge application to receive notifications; subscriptionID is URI used for subscription identifier; filterCriteria is a structure of eventType which defines one or call events of interest (called number, no answer, busy, not reachable), application ID, optionally the called user address; expiryDeadline is time stamp indicating the expiry of the subscription. The accepted subscription is acknowledged by 200 OK response.

The subscriptionID resource represents an existing subscription for call events. The resource supports GET method, PUT method to update an existing subscription, and DELETE method to terminate an existing subscription.

Upon occurrence of call event of interest in the network, the Call Handling mobile edge service sends a POST request to the callback URI provided by the mobile edge application to be notified about the call event. The request body contains eventNotificationInfo data structure indicating the event, the calling user address, and the called user address. The application responds with 200 No Content message which just acknowledges the receipt of notification. After notification, the application uses a POST method to the respective resource to send instructions for call handling (e.g. to the routingRequests resource for call routing, to the transitRequests resource for call continuing, and to the endCallRequets resource for call ending) as described above.

V. Service API Feasibility Study

To assess the practicability of the proposed Call Handling API, we model the call state from network and application points of view. Both state models must expose equivalent behavior i.e. their views on the call state must be synchronized.

The mobile edge application may be triggered on calls to specific service number, or based on the called party state (busy, no answer, unreachable) or user location change. The events on which the application is triggered are regarded as detection points.

Fig.3 shows the simplified outgoing call state model supported by the network. From the network point of view, the call state model exposes well known call states. The proposed extension of the call state model is the adding of a new state where upon an event of interest (detection point), instructions from the mobile edge application are expected.

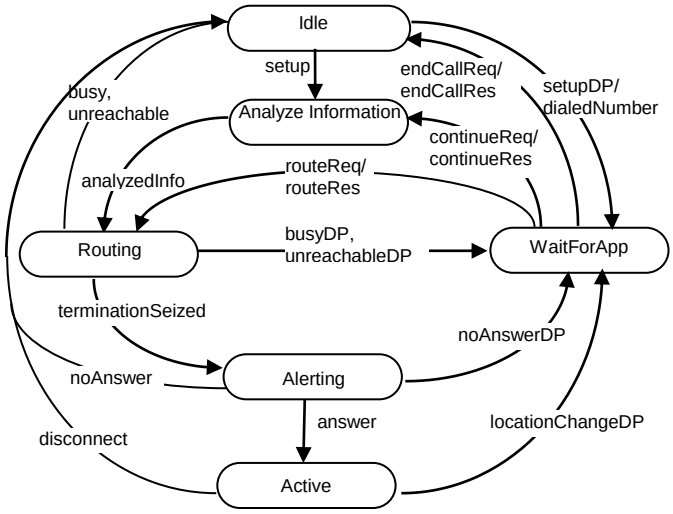

Fig.3 Call state model supported by the network

The call handling continues according to the received application instruction. The model is simplified as it does not consider call abandon from calling party. The semantics of the model states is as follows.

In Idle state, there is no call. Upon a call attempt to specific number, the transition to WaitForApp state occurs. In WaitForApp state, the network waits for instructions from application how to handle the call. In AnalyzeInformation state, the dialed number is analyzed to determine the call routing. In Routing state, the call is routing to is destination. In case of called party is busy or unreachable, the network waits for further call handling instructions from application. In Alerting state, the called party is notified for the incoming call and an answer is awaited. In case of no answer, the application is notified, and the networks waits for its instruction. In Active state, the call is answered, and the communication is ongoing. The calling party may change her/his position and being notified, the application may provide instructions for call handling.

Fig.4 shows the simplified call state model supported by the mobile edge application.

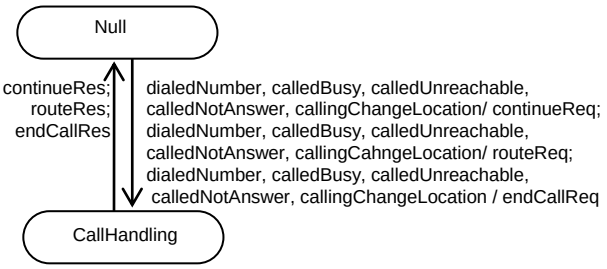

Fig.4 Call state model supported by the application

The call state model is simple as the application can only request the call handling prior to the call being setup and cannot keep control over the call after event handling. In Null state, the application is not activated. The application is activated upon notification about special number dialling, called party busy, called party unreachable, called party does not answer, and change of position. Each event handling is done separately. In CallHandling state, the application waits for the result of its instructions in the network.

Both models are formally described and verified using the concepts of Labeled Transition Systems (LTS) and weak bi-simulation. Both concepts are widely applicable for formal model verification [17].

The outgoing call state model supported by the network is formally described by $T^n = (S^n, Act^n, \rightarrow^n, s_0^n)$, where

- S^n = {Idle [s_1^n], WaitForApp [s_2^n], AnalyzeInformation [s_3^n], Routing [s_4^n], Alerting [s_5^n], Active [s_6^n]};
- Act^n = {setup [t_1^n], setupDP [t_2^n], endCallReq[t_3^n], continueReq [t_4^n], routeReq [t_5^n], analyzeInfo [t_6^n], busyDP [t_7^n], unreachableDP [t_8^n], busy [t_9^n], unreachable [t_{10}^n], teminationSeized [t_{11}^n], noAnswerDP [t_{12}^n], noAnswer[t_{13}^n], answer [t_{14}^n], disconnect [t_{15}^n], locationChangeDP [t_{16}^n]};
- $\rightarrow^n$ = {(s_1^n t_1^n s_3^n), (s_1^n t_2^n s_2^n), (s_2^n t_3^n s_1^n), (s_2^n t_4^n s_3^n), (s_3^n t_6^n s_4^n), (s_2^n t_5^n s_4^n), (s_4^n t_7^n s_2^n), (s_4^n t_8^n s_2^n), (s_4^n t_9^n s_1^n), (s_4^n t_{10}^n s_1^n), (s_4^n t_{11}^n s_5^n), (s_5^n t_{12}^n s_2^n), (s_5^n t_{13}^n s_1^n), (s_5^n t_{14}^n s_6^n), (s_6^n t_{15}^n s_1^n), (s_6^n t_{16}^n s_2^n), };
- $s_0^n = \{s_1^n\}$.

Short notations are given in brackets.

The call state model supported by the application is formally described as $T^a = (S^a, Act^a, \rightarrow^a, s_0^a)$, where:

- S^a = {Null [s_1^a], CallHandling [s_2^a]};
- Act^a = {numberDialed [t_1^a], continueRes [t_2^a], routeRes [t_3^a], endCallRes [t_4^a], calledBusy [t_5^a], calledUnreachable [t_6^a], calledNotAnswer [t_7^a], callingChangeLocation [t_8^a]};
- $\rightarrow^a$ = {(s_1^a t_1^a s_2^a), (s_2^a t_2^a s_1^a), (s_2^a t_3^a s_1^a), (s_2^a t_4^a s_1^a), (s_2^a t_8^a s_1^a), (s_1^a t_5^a s_2^a), (s_1^a t_6^a s_2^a), (s_1^a t_7^a s_2^a), (s_1^a t_8^a s_2^a)};
- $s_0^a = \{s_1^a\}$.

Strong bi-simulation requires strict mapping of both LTSs transitions. In weak bi-simulation, internal transitions in one LTS, which are invisible in the other LTS, can be skipped.

Proposition: T^n and T^a are weakly bi-similar and expose equivalent behavior.

Proof: Let $R \subseteq (S^n \times S^a)$ where $R = \{(s_1^n, s_1^a), (s_2^n, s_2^a)\}$. Then the following mapping between transitions of T^a and T^n can be identified:

1. The application is triggered in case of a call to a specified service number. Upon a call request the network notifies the application: For (s_1^n t_2^n s_2^n) $\exists$ (s_1^a t_1^a s_2^a).

2. The application, which is triggered upon a call to a specified service number, sends an instruction to end the call: For (s_2^a t_4^a s_1^a) $\exists$ (s_2^n t_3^n s_1^n).

3. The application, which is triggered upon a call to a specified service number, sends an instruction to continue the call (to handle the call as transit). The network continues with call setup and the called party may answer the call, or may not answer, or may be busy, or may be unreachable: For (s_2^a t_3^a s_1^a) $\exists$ {(s_2^n t_4^n s_3^n)$\sqcap$ (s_3^n t_6^n s_4^n)$\sqcap$(s_4^n t_{11}^n s_5^n)$\sqcap$(s_5^n t_{14}^n s_6^n)$\sqcap$(s_6^n t_{15}^n s_1^n)}$\sqcup${(s_2^n t_4^n s_3^n)$\sqcap$(s_3^n t_6^n s_4^n)$\sqcap$(s_4^n t_{11}^n s_5^n)$\sqcap$(s_5^n t_{13}^n s_1^n)}$\sqcup$ {(s_2^n t_4^n s_3^n)$\sqcap$(s_3^n t_6^n s_4^n)$\sqcap$(s_4^n t_9^n s_1^n)}$\sqcup${(s_2^n t_4^n s_3^n) (s_3^n t_6^n s_4^n) $\sqcap$(s_4^n t_{10}^n s_1^n)}.

4. The application, which is triggered upon a call to a specified service number, sends an instruction to route the call to the indicated address. The network continues with call setup: For (s_2^a t_3^a s_1^a)$\exists${(s_2^n t_5^n s_4^n)$\sqcap$(s_4^n t_{11}^n s_5^n)$\sqcap$(s_5^n t_{14}^n s_6^n) $\sqcap$(s_6^n t_{15}^n s_1^n)}$\sqcup${(s_2^n t_5^n s_4^n)$\sqcap$(s_4^n t_{11}^n s_5^n)$\sqcap$(s_5^n t_{13}^n s_1^n)} $\sqcup${(s_2^n t_5^n s_4^n)$\sqcap$(s_4^n t_9^n s_1^n)}$\sqcup${(s_2^n t_5^n s_4^n)$\sqcap$(s_3^n t_6^n s_4^n) $\sqcap$ (s_4^n t_{10}^n s_1^n)}.

5. The application, which is triggered upon a call to a specified service number, sends an instruction to end the call. The network terminates the call: for (s_2^a t_4^a s_1^a) $\exists$ (s_2^n t_3^n s_1^n).

6. The application is triggered in case the called party is busy. Upon a call request the network analyses the dialed number, routes the call to the called party, and notifies the application that the called party is busy: For (s_1^a t_5^a s_2^a) $\exists${(s_1^n t_1^n s_3^n)$\sqcap$(s_3^n t_6^n s_4^n)$\sqcap$(s_4^n t_7^n s_2^n)}.

7. The application, which is triggered in case the called party is busy, sends instructions to reroute the call: For (s_2^a t_3^a s_1^a) $\exists${(s_2^n t_5^n s_4^n)$\sqcap$(s_4^n t_{11}^n s_5^n)$\sqcap$(s_5^n t_{14}^n s_6^n)$\sqcap$(s_6^n t_{15}^n s_1^n)}

8. The application, which is triggered in case the called party is busy, sends instructions to end the call: For (s_2^a t_4^a s_1^a) $\exists$(s_2^n t_3^n s_1^n).

9. The application is triggered in case the called party is unreachable. Upon a call request the network analyses the dialed number, routes the call to the called party, and notifies the application that the called party is unreachable: For (s_1^a t_6^a s_2^a) $\exists${(s_1^n t_1^n s_3^n)$\sqcap$(s_3^n t_6^n s_4^n)$\sqcap$(s_4^n t_8^n s_2^n)}.

10. The application, which is triggered in case the called party is unreachable, sends instructions to reroute the call: For (s_2^a t_3^a s_1^a)$\exists${(s_2^n t_5^n s_4^n)$\sqcap$(s_4^n t_{11}^n s_5^n)$\sqcap$(s_5^n t_{14}^n s_6^n)$\sqcap$(s_6^n t_{15}^n s_1^n)}.

11. The application, which is triggered in case the called party is unreachable, sends instructions to end the call: For (s_2^a t_4^a s_1^a) $\exists$(s_2^n t_3^n s_1^n).

12. The application is triggered in case the called party does not answer. Upon a call request the network analyses the dialed number, routes the call to the called party, and notifies the application that the called party does not answer: (s_1^a t_7^a s_2^a) $\exists${(s_1^n t_1^n s_3^n) $\sqcap$(s_3^n t_6^n s_4^n) $\sqcap$(s_4^n t_{11}^n s_5^n) $\sqcap$(s_5^n t_{12}^n s_2^n)}.

13. The application, which is triggered when the called party does not answer sends an instruction to end the call. The network ends the call. For $(s_2^a\ t_4^a\ s_1^a)\ \exists\ (s_2^n\ t_3^n\ s_1^n)$.

14. The application, which is triggered when the called party does not answer, sends an instruction to reroute the call to another destination address. The network reroutes the call. For $(s_2^a\ t_3^a\ s_1^a)\ \exists\{(s_2^n\ t_5^n\ s_4^n)\ (s_4^n\ t_{11}^n\ s_5^n)\sqcap(s_5^n\ t_{14}^n\ s_6^n)\sqcap(s_6^n\ t_{15}^n\ s_1^n)\}$.

15. The application is triggered when the calling party changes her/his position during the conversation. For $(s_1^a\ t_8^a\ s_2^a)\ \exists\{\ (s_3^n\ t_6^n\ s_4^n)\sqcap(s_4^n\ t_{11}^n\ s_5^n)\sqcap(s_5^n\ t_{14}^n\ s_6^n)\sqcap(s_6^n\ t_{16}^n\ s_2^n)\}$.

16. The application, which is triggered upon change of position, sends instructions to transfer the call to another destination address (i.e. to reroute the call). For $(s_2^a\ t_3^a\ s_1^a)\ \exists\ \{(s_2^n\ t_5^n\ s_4^n)\sqcap(s_4^n\ t_{11}^n\ s_5^n)\sqcap(s_5^n\ t_{14}^n\ s_6^n)\sqcap(s_6^n\ t_{15}^n\ s_1^n)\}$

17. The application, which is triggered upon change of position, sends instructions to end the call. For $(s_2^a\ t_4^a\ s_1^a)\ \exists(s_2^n\ t_3^n\ s_1^n)$.

Therefore, T^n and T^a are weakly bi-similar. ■

The bi-simulation is used in system verification and validation. Along with the other approaches, it contributes to increasing software reliability [17].

VI. Estimation of Service Processing Time

In [18], the End-To-End (E2E) latency is defined as a measure for user experience. E2E latency refers to the time taken for a request generated from a device to go to the destination, be processed and replied and travel back to the device [19]. This definition assumes ideal service capabilities e.g. the E2E latency does not depend on server/device computational load. Studies on latency measurements on user plane for MEC-based services are provided in [20], [21]. In [21], network E2E latency is tested and analyzed, and it is conducted that MEC can support services with latency greater than 17 *ms*.

The control plane latency introduced by the proposed MEC service can be evaluated as E2E latency, set-up time, service processing time, and context-update time.

The E2E latency depends on the time required to transmit packets over the radio interface T_R, the time required for backhaul connection between the access and core network T_B, the core network processing time T_C, the time for connection between the core network and MEC server T_T and the service processing time T_S. So, RTT is 2 x (T_R+ T_B+ T_C+T_T+T_S). The proposed service deployment assumes MEC server co-location with distributed core functionality, so the T_T is insignificant. Deployment of distributed core network closer to the edge further reduces the backhaul latency T_B. The Call Handling service processing time is the time taken by the MEC server to process the application or network requests and to generate the respective response. Context-update time must be considered in case of context-aware applications, such as location-dependent call handling.

To assess theoretically the service processing time T_S, we evaluate the time required to process the application requests for call handling and to generate a response based on the call result in the network.

The HTTP request to reroute the call looks like the following:

```
POST /appRootExam/ch/v1/callHandlingRequests/routingReq
uests HTTP/1.1
Host: example.com
Accept: application/json
Content-type: application/json
Content-length: 241

{"actionData":{"actionToEnforce":"route",
"routingAddress":"adf33-12.eerf.bg/bb65",
"mediaInfo":[{"type":"voice","direction":"bidirectional"},{"typ
e":"video","direction":"bidirectional"}],"appID":"2143ABF15",
"requestID: "75EE14-B67-99C"}}
```

The corresponding HTTP response looks as:

```
HTTP/1.1 201 Created
Location:/appRootExam/ch/v1/callHandlingRequests/routing
Requests/2233ABF15
Content-type: application/json
Content-length: 241

{"actionData":{"actionToEnforce":"route","routingAddress":"
adf33-12.eerf.bg/bb65",
"mediaInfo":[{"type":"voice","direction":"bidirectional"},{"typ
e":"video","direction":"bidirectional"}],"appID":"2143ABF15",
"requestID: "75EE14-B67-99C"}}
```

The MEC service processing time T_S can be defined as a product of the input task size (i.e. information block size in bits), the complexity of the input task which reflects the necessary CPU cycles per bit, and the MEC's server CPU frequency. So, the service processing time T_S for rerouting a call, is evaluated using the following parameters: information size of 6360 bits (406 symbols for the request and 389 symbols for the response); 1000 cycles per bit for complexity; and CPU frequency at 2.2 GHz, which gives the time budget of 2.89 *ms*.

The experimental data in [22] aim an assessment of MEC latency, where the authors use fiber-wireless access in order to provide E2E latency measurement setup. To isolate any application specific processing from the latency assessment in [22], the experiment is made only using ping, while we add a summand representing the theoretically evaluated MEC service processing time. Then, using data of the experiment as empirical basis, we try to match an appropriate distribution. The set of candidate distributions for fitting the empirical cumulative distribution function (ECDF), which is solid-black in Fig.5, consists of the well-known ones like LogNormal, Normal, Gamma, and Weibull.

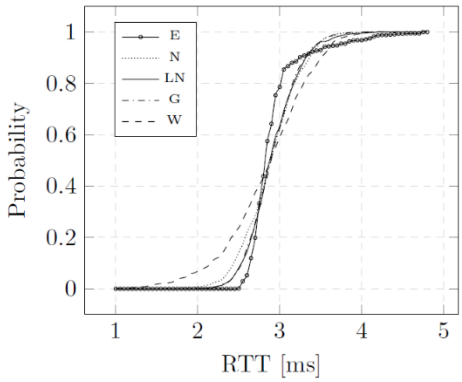

Fig.5 Theoretical and empirical and CDFs of RTT as KPI for latency in call rerouting by MEC application

The capital letters in Fig.5 and here annotate the RTT CDF (Cumulative Distribution Function) for the Empirical one, for Weibull ($k = 6.446$; $\lambda = 3.073$), Gamma ($\alpha = 82.606$; $\beta = 28.421$), LogNormal ($\mu = 1.061$; $\sigma = 0.106$), and Normal ($\mu = 2.907$; $\sigma = 0.348$) respectively.

The comparison of the empirical distribution to the fitted ones is shown in Table I, where the greater values mean greater 'mismatch' with respect to the original data. The well-known statistics like Anderson-Darling, Cramer-von Mises, and Kolmogorov-Smirnov are abbreviated to capital letters in Table I.

TABLE I. GOODNESS-OF-FIT STATISTICS

	KS	**CvM**	**AD**
Gamma	0.18921410	9.8152310	55.9899720
LogNormal	0.19362510	8.9214432	50.1021147
Normal	0.22882951	12.0965092	68.1143217
Weibull	0.25267756	19.6115671	103.3287692

The goodness-of-fit statistics are the necessary instrument when comparing different fits to raw data and come to be base for the selection procedure of the best fit done. The closest candidate distribution fit is the LogNormal one as she has lowest values at all types of statistics.

The resulting distribution might be used when one must model latency compactly instead of keeping in cache considerable amount of the original dataset. For example, when it is needed to estimate the injected latency by the software component, which is responsible for interface implementation in the application or service development process, the developer can draw samples from the statistical model and thus to follow the respective quality of service requirements regarding the latency budget.

VII. CONCLUSION

The open access to call handling functionality enables third party applications deployed at the network edge to determine flexibly how the calls should be treated. The proposed functionality enables mission critical applications at the end to react appropriately in different call scenarios like special number dialing, no answer, busy, no reachable or change of position. Based on the specific situation or analytics, the applications may reroute the call, terminated the call or transfer the call.

The main benefits of the proposed approach to deploy mission critical voice applications at the 5G network edge may be summarized as follows. First, in contrast to MCPTT service, the proposed mobile edge service provides full duplex communications and does not require deployment of IMS at the network edge which reduces signaling. Second, due to vicinity of end users, many use cases with requirements of low latency and high reliability may be supported e.g. mission critical sessions and real-time IoT applications. Third, the proposed functionality enables mobile edge applications to react rapidly and flexibly in situation where timely response is critical. Last, but not least, moving the call handling control at the edge saves backhaul resources and it is useful in cases where the backhaul connection is lost as all interactions are executed locally.

Possible improvements of the proposed approach related to future work include typical for telecommunications use cases feedback with the calling party where notifications to the calling party about application decisions on call handling are sent. For mission critical voice calls this function is not mandatory as the aim is to responds as quickly as possible. For the purposes of mission critical communications, handling of other media types also may be included such as video, data, text. Any kind of sensor information e.g. the radiation level, temperature or smoke may be transferred along with the voice, so the session handling at the edge may also be useful.

References

[1] N. A. Mohammed, A. M. Mansoor and R. B. Ahmad, "Mission-Critical Machine-Type Communication: An Overview and Perspectives Towards 5G," in IEEE Access, vol. 7, pp. 127198-127216, 2019.

[2] H. Wang, Q. Yang, Z. Ding and H. V. Poor, "Secure Short-Packet Communications for Mission-Critical IoT Applications," in IEEE Transactions on Wireless Communications, vol. 18, no. 5, pp. 2565-2578, May 2019.

[3] J. Sachs, G. Wikstrom, T. Dudda, R. Baldemair and K. Kittichokechai, "5G Radio Network Design for Ultra-Reliable Low-Latency Communication," in IEEE Network, vol. 32, no. 2, pp. 24-31, March-April 2018.

[4] K. Williams, M. Assaf, Intelligent Public Transportation, International Journal of Mathematics and Computers in Simulation, vol.12, 2018, pp.124-132.

[5] E. Jimeno, J. Pérez-Romero, I. V. Muñoz, B. Blanco, A. Sanchoyerto and J. F. Hidalgo, "5G Framework for automated network adaptation in Mission Critical Services," 2018 IEEE Conference NFV-SDN, Verona, Italy, 2018, pp. 1-5.

[6] Q. Zhang, F.H.P. Fitzek. Mission Critical IoT Communication in 5G. In: V. Atanasovski, A. Leon-Garcia (eds) Future Access Enablers for Ubiquitous and Intelligent Infrastructures. FABULOUS 2015.

Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 159. Springer, Cham.
[7] S. Mukherjee and C. Beard, "A framework for ultra-reliable low latency mission-critical communication," 2017 Wireless Telecommunications Symposium (WTS), Chicago, IL, 2017, pp. 1-5.
[8] T. Yang, X. Shen. Mission-Critical Search and Rescue Networking Based on Multi-agent Cooperative Communication. In: Mission-Critical Application Driven Intelligent Maritime Networks, 2020 SpringerBriefs in Computer Science. Springer, Singapore.
[9] A.A.R. Alsaeedy, E.K.P. Chong, 5G and UAVs for Mission-Critical Communications: Swift Network Recovery for Search-and-Rescue Operations. Mobile Netw Appl, 2020.
[10] M. Höyhtyä *et al*., "Critical Communications Over Mobile Operators' Networks: 5G Use Cases Enabled by Licensed Spectrum Sharing, Network Slicing and QoS Control," in IEEE Access, vol. 6, pp. 73572-73582, 2018.
[11] E. Pencheva, I. Atanasov, V. Vladislavov. "Mission Critical Messaging Using Multi-Access Edge Computing," Cybernetics and Information Technology, vol.19, no.4, 2019, pp.73-89.
[12] R. Solozabal, A. Sanchoyerto, E. Atxutegi, B. Blanco, J. O. Fajardo and F. Liberal, "Exploitation of Mobile Edge Computing in 5G Distributed Mission-Critical Push-to-Talk Service Deployment," in IEEE Access, vol. 6, pp. 37665-37675, 2018.
[13] A. Sanchoyerto, R. Solozabal, B. Blanco and F. Liberal, "Analysis of the Impact of the Evolution Toward 5G Architectures on Mission Critical Push-to-Talk Services," in IEEE Access, vol. 7, pp. 115052-115061, 2019.
[14] F. Giust, et al. "MEC Deployment in 4G and Evaluation Towards 5G," ETSI white paper no.24, pp.1-24, 2019.
[15] 3GPP TS 29.513 Technical Specification Group Core Network and Terminals; System Architecture for the 5G System (5GS), Stage 2, Release 16, v16.3.0, 2019
[16] ETSI GS MEC 009 Mobile Edge Computing (MEC); General principles for Mobile Edge Service APIs, v1.1.1, 2017.
[17] J. Zhang, Y. Lu, K. Shi, C. Hu, Applying Software Metrics to RNN for Early Reliability Evaluation, International Journal of Mathematical Models and Methods in Applied Sciences, vol.13, 2019, pp. 96-102
[18] NGMN Alliance 5G White Paper version 1.0 (17 February 2015): "NGMN 5G White Paper".
[19] ETSI GS MEC-IEG 006; "Mobile Edge Computing; Market Acceleration; MEC Metrics Best Practice and Guidelines, " v1.1.1, 2017
[20] M. Emara, M. Filippou, D. Sabella. "MEC-assisted End-to-End Latency Evaluations for C-V2X Communications," EuCNC'18, Cornell University, arXiv:1802.08027 [eess.SP], 2018, pp.1-5.
[21] J. Zhang, W. Xie, F. Yang and Q. Bi, "Mobile edge computing and field trial results for 5G low latency scenario," in China Communications, vol. 13, no. Supplement2, pp. 174-182, 2016.
[22] J. Liu, G. Shou, Y. Liu, Y. Hu and Z. Guo. "Performance Evaluation of Integrated Multi-Access Edge Computing and Fiber-Wireless Access Networks," IEEE Access, vol. 6, pp. 30269-30279, 2018.

Ivaylo Atanasov is born in Sofia, Bulgaria. He has received his MSc degree in Electronics and PhD degree in Communication networks from Technical University of Sofia (TU-Sofia). He has been awarded DSc degree in communication networks in 2016 from the Faculty of Telecommunications, TU-Sofia. Since 2013, he is Professor and his scientific research area covers mobile networks, internet communications and protocols, and mobile applications.

Evelina Pencheva is with the Faculty of Telecommunications, Technical University of Sofia. She is born in Sofia. She has received her MSc degree in Mathematics from Sofia University "St. Kliment Ohridski" and PhD degree in Communication networks from TU-Sofia. She has defended her DSc thesis in 2014. Since 2010, she is Professor and her scientific research area covers multimedia networks, telecommunication protocols, and service platforms.

Aleksander Nametkov is born in Sofia, Bulgaria in 1989. He has received the MSc. degree is received at the TU-Sofia in 2015. The area of expertise is in Communication Networks and Safety Critical Systems. He is currently working on his PhD thesis (start at 2019) with Faculty of Telecommunications, TU-Sofia. His research interests include mobile networks and services, artificial intelligence, and intrusion detection.