Media Control at the Network Edge

Evelina Pencheva, Ivaylo Atanasov
Technical University of Sofia
Sofia, Bulgaria
enp,iia@tu-sofia.bg

Denitsa Velkova, Ivaylo Asenov
Technical University of Sofia
Sofia, Bulgaria
denitsa.velkova, ivaylo.asenov@balkantel.net

***Abstract*—Third party session control can be useful in many Internet of Things (IoT) use cases including monitoring and assistance of elderly and mission critical communications. Enhanced session control with media playing, collecting and recording interaction adds value and enables more information to be provided and gathered. The Multi-access Edge Computing (MEC), which brings the cloud computational and storage power at the network edge, can address many IoT use cases. The paper studies the capabilities of exposing media control to MEC applications. MEC Application Programming Interfaces that enable third party application to play media message, to collect and record media interactions within ongoing session, are presented. Media control data model is proposed. As a proof of the concept, state models supported by the network and MEC application are presented, formally described and verified.**

I. Introduction

With the ubiquitous penetration of Internet of Things (IoT) devices, a lot of new services that capture a great attention continuously become available in different areas such as healthcare, home automation, intelligent location tracking, entertainment etc. [1], [2]. IoT devices are usually constrained with respect to storage and processing capabilities and sometimes power supply and network resources, while the services are computation and storage intensive. The centralized cloud services can provide the required resources, but the latency introduced is unacceptable for services with real-time requirements such as industrial robots, telemedicine, connected cars etc. [3], [4]. The emergence of Multi-access Edge Computing (MEC) addresses the latency issues by filling the gap between centralized clouds and IoT devices. MEC enables offloading of computing load and provisioning of data storage capabilities at the network edge, closer to where they are needed [5], [6]. The distributed computing environment provided by MEC facilitates low latency and ultra-reliable services. MEC empowers IoT applications using machine learning techniques to retrieve insights of data gathered by sensors and thus to provide better services. The offloading of heavy tasks from IoT devices to MEC server mitigates the communication and computational overhead. Further, MEC can contribute to network traffic congestion alleviation by preprocessing data of massive IoT applications [7], [8].

Currently standardized MEC services provide up-to-date radio network and location information and enable traffic steering between networks, services and applications. MEC computing environment may be beneficial for telecom operators but also for third party service providers. The MEC potential to provide open access to network functions is beyond existing functionality. MEC can provide computing environment for applications that may use of core network functions like messaging, call control and multicasting exposed for third parties, [9], [10], [11].

In this paper, we study the MEC capabilities to provide applications with media control functionality. The research is adopting the idea of Parlay X Audio Call web service [12]. The Audio Call web service enables applications to send multimedia messages in the context of ongoing call as well as to manage media types for participants involved in a call. The current research is focused on distribution of these capabilities at the network edge following the Representational State Transfer (REST) architectural style, adopted by European Telecommunication Standard Institute (ETSI) for MEC services. Contrary to Parlay X Audio Call, the access to media type management for participants involved in a session is a part of functionality of other MEC service which provides REST-based interfaces for third-party session control, including dynamic management of session participants.

The rest of the paper presents the MEC deployment scenario for media control, use cases illustrating the benefits of media control in the vicinity of end users, data model and respective Application Programming Interfaces (APIs) for media control as well as service state models supported by the network and applications.

II. Deployment of Media Control at the Edge of the Mobile Network

As to MEC reference architecture, the MEC server hosts mobile edge platform and Network Function Virtualization (NFV) platform that provides cloud and communication resources. The basic functionalities required for applications to run are provided by the mobile edge platform. MEC applications are running as Virtualized Network Functions (VNF) on the top of NFV platform.

For static IoT devices where mobility management is not required, the mobile edge platform hosting IoT applications (e.g. for video surveillance, industrial control, home automation) can be deployed both at the radio access node and at aggregation point close to the radio access network [13].

For moving IoT devices where mobility management and service continuity is required (e.g. vehicles, drones, trains) the mobile edge platform can be co-located with distributed core network functions. In this case, core network functions AMF (Access and mobility Management Function), SMF (Session Management Function), PCF (Policy Control Function), which are responsible for mobility management and session

management, UPF (User Plane Functions) that handles user traffic, and UDR (User Data Repository) are distributed at the network edge to face the performance issues associated with the centralized control [14]. The distributed core network architecture is suitable for mission-critical communications and IoT scenarios, where the communication with the operator's core functionality is optional [15], [16]. It enables provisioning of required quality of service (QoS) and customized features configuration. Furthermore, when distributed core network functions are virtualized as VNFs, mobile edge applications can run on the same virtualized platform as part of the same MEC server. This deployment improves scalability and enables more efficient resource usage.

The MEC deployment with distributed virtualized core network functions is shown in Fig.1. This scenario serves well session management and policy control functionality required for media control at the network edge.

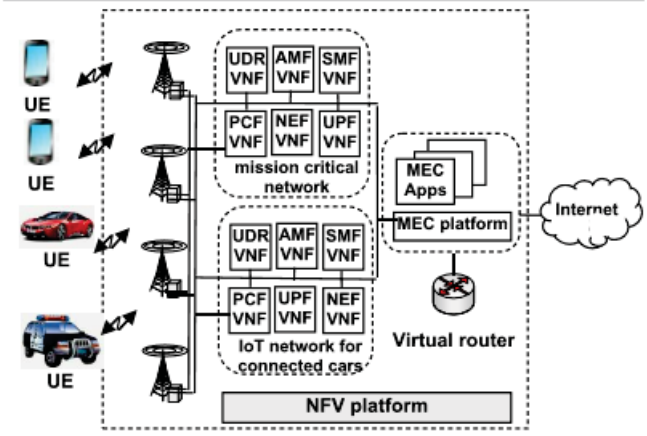

Fig.1 MEC deployment with distributed virtualized core network functions

Secured exposure of core network capabilities and events is provided by NEF (Network Exposure Function) [17]. The NEF also provides secure information from external applications to the network and packet flow description management. The NEF is responsible for authorization of all requests originating from MEC applications.

The exposed capabilities which can be accessed by the mobile edge platform through the NEF include the following:

- Monitoring of User Equipment (UE) related events,
- Provisioning of prognostic information about UE behavior,
- Handling QoS policy for UE based on application's requests.

The mobile edge platform can use the exposed capabilities to provide MEC applications with media control functionality.

The proposed media control APIs enable MEC applications to play media, to capture and record for session participants. Capabilities for media playing include playing of text, audio and video messages, retrieval of message status and terminating the message playing. The capabilities for media capturing can be used by MEC applications to play a media file to session participant(s) and to collect information from session participant(s), to stop media interaction, as well as to record interaction.

The functionality for third party session management and adding or removing of session participants, and dynamic media type management for session participants is exposed by APIs for third-party session control, but it is not discussed here.

Next section illustrates the basic Media Control API functionality by typical use cases.

III. Description of Media Control API Functionality

Fig.2 illustrates the structure of the resource URIs supported by the proposed API.

All resource URIs follow a common root that can be discovered using service registry. The APIs support information about all requests for media messages to be played to session participants, and all interaction requests with session participants. All resources are manipulated using four simple operations mapped onto HTTP methods – CREATE (HTTP POST), READ (HTTP GET), UPDATE (HTTP PATCH) and DELETE (HTTP DELETE).

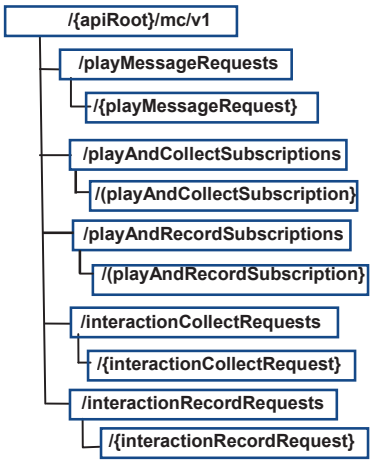

Fig.2 Structure of media control resource URIs

A. Message Playing

A use case of media playing is as follows. The application tracks the location of a child and finds that the child is not in the area (e.g. school area) where it is expected to be. The application initiates a session with the parent and using the media playing API she plays an audio message to the parent.

Another use case is video surveillance in home automation. The application detects intrusion, initiates a session between the homeowner and the security officer, and using the media control API sends a video stream of the intrusion.

The playMessageRequests resource represents all requests for media messages to be delivered for ongoing session. The

messages to be played to the session participants can be in different media forms including text, audio and video.

When a MEC application wants a media message to be played, she sends a POST request to the playMessageRequests resource. The request body contains MediaMessageInfo data structure which specifies the session identifier, session participants, message information and optionally charging information, and the application instance identifier. The application allocates a unique request identifier for subsequent interactions. The media control service responds with "201 Created" and response body containing media message information specific for that request.

Fig.3 shows the flow for a request of playing media message. A request for media message playing results in execution of Application Function (AF) influence on traffic routing procedure described in [14]. According to this procedure, the MEC platform, in a role of AF, can send requests to influence SMF routing decisions for traffic of packet sessions. The MEC platform uses NEF to interact with 5G core network. On application request for message playing, the media control service invokes Nnef_TrafficInfluence_Create operation of the Nnef_TrafficInfluence service as described in [18], [19]. The Nnef_TrafficInfluence service authorizes the request, forwards the request for traffic influence and creates subscription for traffic related events. On receiving operation execution result indication, the media control service responds to the application.

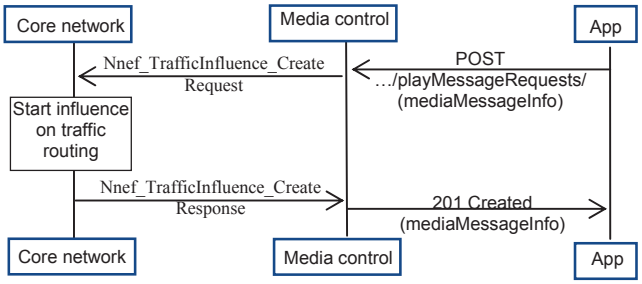

Fig.3 Flow for a request of media message playing

The playMessageRequest resource represents an existing request for message playing. The application can retrieve the media message status and cancel a previous message playing request.

As with the traffic influence request, the media control service has created a subscription for traffic related events, it received notifications from the NEF for message playing status. Fig.4 shows the flow for retrieval of message status by the application.

In order to retrieve the message status, the application sends a GET requests to the resource representing the request for message playing with message body including the message correlator. The response contains the respective message status which can be one of the following: played (the message has been played), playing (the message is currently playing), pending (the message has not started yet) and error (the message will not be played as an error has occurred).

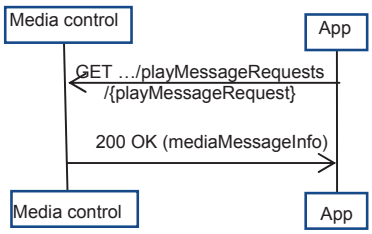

Fig.4 Flow for requests for message playing status

The MEC application can change the request for message playing. To do this, she uses REST based procedure by sending PUT or PATCH request to the resource representing the request for message playing. The request body contains updated MediaMessageInfo data structure. The media control service in turn invokes Nnef_TrafficInfluence_Update operation of the Nnef_TrafficInfluence service. This operation authorizes the request and forwards it to update the traffic influence. On receiving indication about the result of executed operation, the media control service responds to the MEC application with "200 OK".

The flow for updating the request for message playing is shown in Fig.5.

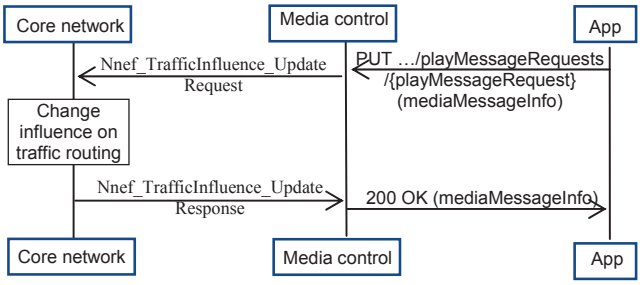

Fig.5 Flow for updating the request of media message playing

Fig.6 shows the flow for message playing cancelation. When the application wants to cancel the message playing previously requested, she sends a DELETE request to the resource representing the request for message playing. This results in execution of Nnef_TrafficInfluence_Delete operation of the Nnef_TrafficInfluence service.

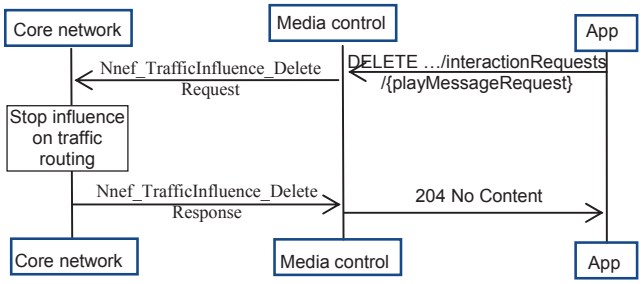

Fig.6 Flow for a request for cancelation of media playing.

B. Media Interaction

Following the location tracking use case, the application using media control API can start an interaction with the parent asking him whether to initiate a session with the child. Another use case is an analytic application that detects elder people's bad mood caused by stress or troubles, initiates a session with the stressed person and using media control API starts a video stream to cheer him/her up. Typical use cases of media interactions in control of communicable diseases is provided in [20]. Use cases and requirements for media recording can be found in [21].

In order to capture or record media from session participants, the application creates a subscription for notifications about events related to interactions with participants. The playAndCollectSubscriptions resource represents all subscriptions for notifications related to media capturing related events in the context of interaction with session participant(s). The playAndRecordSubscriptions resource represents all subscriptions for notifications related to recoding of media provided by session participant(s).

When the application wants to capture media from session participant, it sends a HTTP POST requests to the playAndCollectSubscriptions resource with message body containing PlayAndCollectSubscription data structure. The PlayAndCollectSubscription data type contains the session ID, session participant URIs and the callback address where the application wants to receive notifications. Upon receiving the request, the media control service invokes Nnef_EventExposure_Subscribe operation of the Nnef_EventExposure service to subscribe for receiving notifications about interaction related events. When the subscription is accepted, the media control service responds to the application with “201 Created” and response body containing data structure specific to that interaction related subscription.

Fig.7 shows a scenario where the application uses REST-based procedure to create a subscription for notifications about media captured.

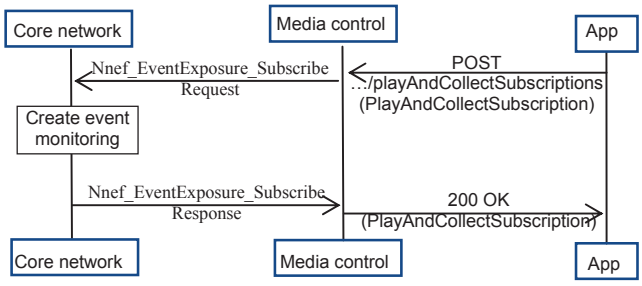

Fig.7 Flow of subscribing to media capture event notifications

The application can update the subscription (e.g. adding/ removing a session participant to/from interaction). When the application needs to modify existing subscription for interaction related events, she sends a PUT request to the resource representing the respective subscription with a message body containing data specific to the updated subscription. Upon receiving the request, the media control service invokes Nnef_EventExposure_Subscribe operation of the Nnef_EventExposure service to update the existing subscription for receiving notifications about interaction related events. When the subscription update is accepted, the media control service responds to the application with “200 OK” response body containing data structure specific to that interaction related subscription. The media control service returns to the application “200 OK” with the message body containing the accepted data structure specific for that event subscription.

When the application does not want to receive notifications about media interaction anymore, she terminates the subscription by sending a DELETE request to the resource representing the respective subscription. The media control service invokes Nnef_EventExposure_Unsubscribe service operation of the Nnef_EventExposure service to terminate the subscription. Upon receiving indication about operation execution result, the media control service responds to the application.

Fig.8 shows the flow of successful subscription termination for notifications about media capture events.

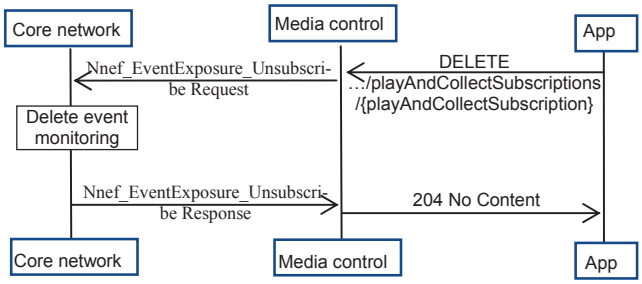

Fig.8 Flow of unsubscribing from media capture event notifications

The interactionCollectRequests resource represents all requests for media interactions with session participants. The application can initiate a new media interaction in which she plays an announcement to session participants and collects information from a session participant.

When a MEC application wants to start playing an announcement and collect information from session participants, she sends a POST request to the interactionRequests resource. The request body contains MediaInteractionInfo data structure which specifies the session identifier, session participants, playing configuration, collected information configuration, and the application instance identifier. The media control service responds with “201 Created” with response body containing media interaction information specific for that request and the interaction correlator for subsequent interactions.

The flow for a request of media interaction with session participants is shown in Fig.9.

A request for media interaction results in execution of Nnef_TrafficInfluence_Create operation of the Nnef_Traffic-Influence service.

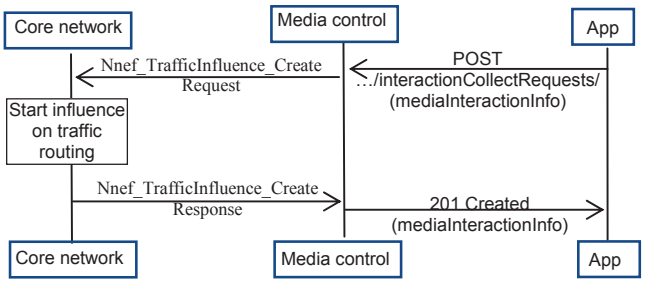

Fig.9 Flow for a request of media interaction with session participants

In case of active subscription for notifications about media captured events, the media control service notifies the application about information captured from the participant. Fig.10 presents a scenario where the media control service uses REST based procedure to send notification on media capture event to the application. When the NEF needs to report the media capture event to the media control service that has previously subscribed, it invokes Nnef_EventExposure_Notify service operation. The media control service in turn sends a POST request with message body containing the MediaCapturedNotification data structure to the callback address provided by the application. The application responds with a "204 No Content".

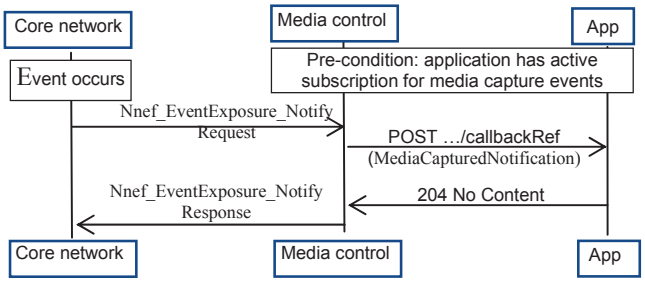

Fig.10 Flow of receiving notifications on media capture event notifications

The interactionCollectRequest resource represents existing interaction request to capture media from session participant.

The application can update the ongoing interaction with a session participant using REST based procedure. To do this, the application sends a PUT or PATCH request to the resource representing the respective interaction. A request for media interaction update results in execution of Nnef_TrafficInfluence_Update operation of the Nnef_TrafficInfluence service. On receiving indication about the result of executed operation, the media control service responds to the MEC application with "200 OK".

The application can terminate the ongoing interaction with a session participant. To do this, she sends a DELETE request to the resource representing the respective interaction. The media control service responds with "204 No Content" as shown in Fig.11.

A request to cancel media interaction results in execution of Nnef_TrafficInfluence_Delete operation of the Nnef_TrafficInfluence service.

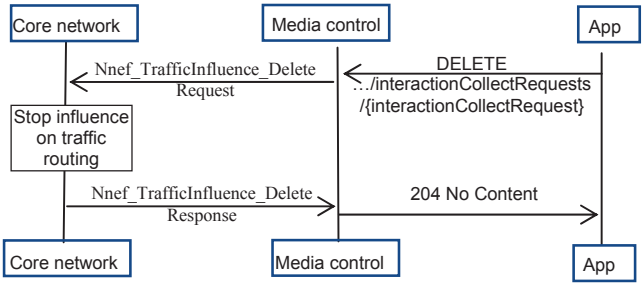

Fig.11 Flow of cancelation of media interaction with session participants

The application can play a media message to one or more session participants on an existing session and record media from a session participant. Media recording may be initiated by a contact center customer service application or financial trading desks. Typical use cases include media recording for mission critical services. In some use cases, media recording is required to resolve user complaints by inspecting visual session replays or to fit out the sessions of premium customers in order they receive enhanced service.

Following the same pattern of communication, the application can subscribe for and unsubscribe from notifications about media interaction recoding. The application can send a POST request to the interactionRecordRequests resource to play a media file to session participant(s) and record media information from a session participant. The request body contains MediaRecordInfo data structure which specifies the session identifier, session participants, playing configuration, recording configuration, and the application instance identifier.

The interactionRecordRequest resource represents existing request for media playing and recording. The application can interrupt the ongoing media playing and recording by using REST based procedure.

IV. Data Model and API Definition

The data model defines the data types and structures related to media control.

The MediaMessageInfo data type defines the details of message to be played to session participant(s). It is a structure of timestamp, sessionID, sessionParticipants, messageType, ChargingInfo, appID and requestID where:

- timeStamp is time stamp,
- sessionID is of string type and identifies the session to which the message must be played,
- sessionParticipants is a structure of one or more participant addresses within the session to which the message has to be played,
- messageType is a structure of mediaType and messageContent. The mediaType is of enumeration type (text, audio, video). The messageContent type defines the text to process with Text-to-Speech engine for text messages, or the URI indication the

location of the audio or video content to be played for audio or video messages,

- chargingInfo defines the charge to apply for message playing,
- appID is of string type and is unique identifier for the MEC application,
- requestID is of string type and is unique identifier allocated by the MEC application for message playing request.

The PlayAndCollectSubscription data type represents a subscription to inputs captured from session participant(s). It is a structure of callbackRef, subscriptionID, filterCriteriaCollect and expiryDeadline, where:

- callbackRef is URI selected by the MEC application to receive notifications on collected inputs from a session participant,
- subscriptionID is self-referring URI regarded as subscription identifier,
- filterCriteria is a structure of collectParticipants which defines one or more session participants from whose media has to be captured, represented by their URIs, appID as defined above, and one or more identifiers to associate the information for a specific participant,
- expiryDeadline is time stamp indicating the subscription expiry.

The MediaInteractionInfo data type contains information about media interaction. It is a structure of timesStamp, sessionID, sessionParticipants, playParameters, captureParameters, appID, and requestID where:

- sessionID and sessionParticipants represent the session and session participants involved in the interaction,
- playParameters represents the playing configuration and it is a structure of fileLocation which is URI pointing to the location of the file that has to be played, text which is a string representing the text to be converted by Text-to-Speech engine, mediaType indicating the media type of the announcement to be played, and interruptMedia which is of Boolean type and indicates whether the participant is allowed to interrupt, pause the prompt;
- captureParameters is of string type and it represents the configuration of the input capture from the participant,
- pauseMedia indicates whether the session participant is allowed to pause or interrupt the prompt.

The MediaCaptureNotification data type represents the input captured from session participant(s). It is a structure of subscriptionID as defined earlier, collectParticipant data type which is a structure of one or more session participants and collectedInfo from each of them.

The PlayAndRecordSubscription data type represents a subscription for notifications on events related to retrieving media from session participant(s). It is a structure of callbackRef, subscriptionID, expiryDeadline, as described above and filterCriteriaRecord. The filterCriteriaRecord data type is a structure of recordParticipants which defines one or more session participants whose media must be recorded, represented by their URIs, appID as defined above, and one or more identifiers to associate the media for a specific participant.

The MediaRecordNotification data type represents the media recorded from session participant(s). It is a structure of subscriptionID as defined earlier, recordParticipant data type which is a structure of one or more session participants and recordedInfo from each one.

The MediaRecordInfo data type contains information about recorded media. It is a structure of timeStamp. sessionID, sessionParticipants, playParameters, recordParameters, appID, and requestID where recordParameters define configuration parameters related to media recording, containing the location for storing the recorded media, and maxRecortingTime representing the maximum time to record the media.

Table I summarizes the resources of media control service and supported methods.

TABLE I. RESOURCES OF MEDIA CONTROL SERVICE AND SUPPORTED METHODS

Resource name	Resource URI	HTTP method	Meaning
All requests for message playing	/playMessageRequests	GET POST	Retrieves the list of requests for message playing Creates a new request for message playing
An existing request for message playing	/playMessageRequests /{playMessageRequest}	GET PUT DETELE	Retrieves information about existing request Modifies existing request Cancels existing request
All subscriptions for notifications on media capturing	/subscriptions /playAndCollectSubscriptions	GET POST	Retrieves list of all subscriptions for notifications on media capture Creates a new subscription
Existing subscription for notifycations on media capturing	/subscriptions /playAndCollectSubscriptions/ {playAndCollectSubscription}	GET PUT DELETE	Retrieves information about the subscription Update the subscription Terminates the subscription
All subscriptions for notifications on media recording	/subscriptions /playAndRecordSubscriptions	GET POST	Retrieves list of all subscriptions for notifications on media recording Creates a new subscription
Existing subscription for notifycations on media recording	/subscriptions /playAndCollectSubscriptions/ {playAndRecordSubscription}	GET PUT DELETE	Retrieves information about the subscription Update existing subscription Terminates the subscription

TABLE I CONTINUE

Resource name	Resource URI	HTTP method	Meaning
All requests for playing an announcement and collecting information	/interactionCollectRequests	GET POST	Retrieves the list of requests for play and collect information Creates a new request
Existing request for playing an announcement and collecting information	/interactionCollectRequests /{interactionCollectRequest}	GET PUT DELETE	Retrieves information about existing request Modifies existing request Cancels existing request
All requests for playing an announcement and recording media	/interactionRecordRequests	GET POST	Retrieves the list of requests for play and record media Creates a new request
Existing request for playing an announcement and recording media	/interactionRecordRequests /{interactionRecordequest}	GET PUT DELETE	Retrieves information about existing request Modifies existing request Cancels existing request

V. STATE MODELS

As a proof of the concept, models representing the message playing status as seen by the mobile edge application and the MEC server are designed. The models must be synchronized, i.e. to expose equivalent behaviour.

In this section, models representing the message playing status are proposed and formally described using the concept of Labelled Transition Systems (LTS). The formal model description enables a mathematical proof that they have a bi-simulation relation e.g. expose equivalent behaviour.

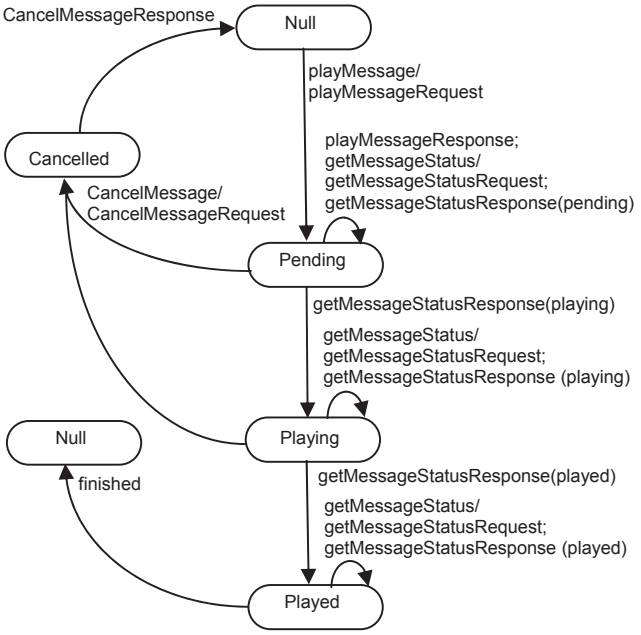

Fig.12 Message status as seen by the application

Fig.12 shows an application view on the message status. The model is simplified as far as it does not consider the exceptional situations with message playing.

The message status from application point of view may be Null, Pending, Playing or Played. The status is Null when there is no message to be played. The status is Pending if the message is waiting to be delivered to the network. The status is Playing or Played if the message playing is ongoing or terminated respectively. The MEC application may cancel message playing.

By $T_{App} = (S_{App}, Act_{App}, \rightarrow_{App}, s_0^{App})$ it is denoted an LTS representing the application view on the message status where:

- S_{App} = {Null [s^A_1], Pending [s^A_2], Playing [s^A_3], Played [s^A_4], Cancelled [s^A_5]};
- Act_{App} = { playMessage [t^A_1], palyMessageResponse [t^A_2], getMessageStatus [t^A_3], getMessageStatusResponse(pending) [t^A_4], getMessageStatusResponse(playing) [t^A_5], getMessageStatusResponse(played) [t^A_6], cancelMessage [t^A_7], cancelMessageResponse [t^A_8],finished [t^A_9]};
- $\rightarrow_{App}$ = { (s^A_1 t^A_1 s^A_2), (s^A_2 t^A_2 s^A_2), (s^A_2 t^A_3 s^A_2), (s^A_2 t^A_4 s^A_2), (s^A_2 t^A_5 s^A_3), (s^A_3 t^A_3 s^A_3), (s^A_3 t^A_5 s^A_3), (s^A_3 t^A_6 s^A_4), (s^A_4 t^A_3 s^A_4), (s^A_4 t^A_6 s^A_4), (s^A_4 t^A_9 s^A_1), (s^A_3 t^A_7 s^A_5), (s^A_2 t^A_7 s^A_5), (s^A_5 t^A_8 s^A_1)}
- s_0^{App} = {s^A_1}.

Notations in brackets are for short names.

Fig.13 shows the simplified model of the message state as seen from the media control service point of view. In Idle state, there is no message to be played.

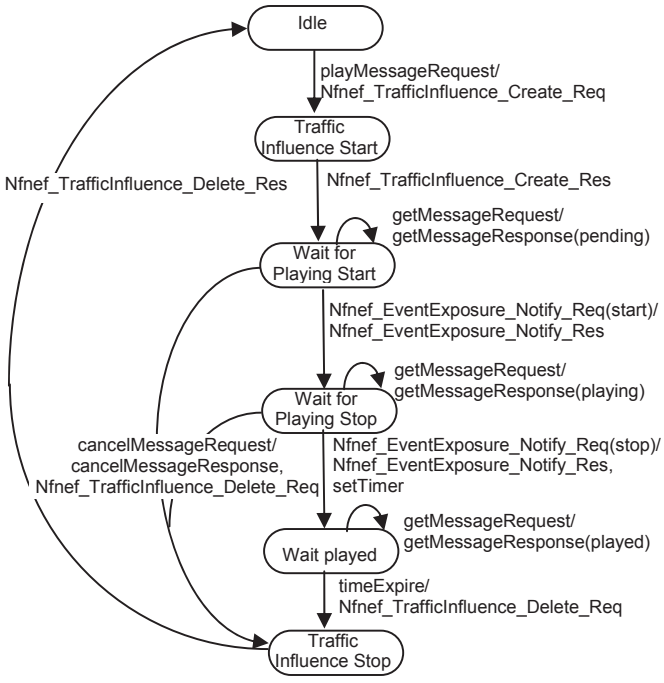

Fig.13 The message state as seen by the MEC server

On receiving a request from an application to play a message, the media control service invokes the Nnef_TrafficInfluence_Create operation, which requests to influence SMF routing decisions for user plane traffic of packet session and creates a subscription for SMF events, and

the message playing status becomes TrafficInfluenceStart. Upon indication of successful operation execution, the message playing state is occupied, in which notification about SMF event for message playing start is awaited. The message playing status in WaitForPlayingStart state is pending and the status becomes Playing on notification about message playing start event. In WaitForPlayingStop state, the message playing status is played until a notification about message playing stop event is received. In WaitPlayed state, the media control service maintains the message playing status for some time after which the message status is released and the Nnef_TrafficInfluence_Delete operation is invoked. In TrafficInfluenceStop state, the request for traffic influence delete is processed in the network. The application may query about message status in WaitFor PlayingStart state, WaitFor-PlayingStop state and WaitPlayed state. The application may cancel the message playing in WaitFor PlayingStart state and in WaitForPlayingStop state.

By $T_{Mec} = (S_{Mec}, Act_{Mec}, \rightarrow_{Mec}, s_0^{Mec})$ it is denoted an LTS representing the MEC server view on the message status where:

- S_{Mec} = {Idle [s^M_1], TrafficInfluenceStart [s^M_2], WaitForPlayingStart [s^M_3], WaitForPlayingStop [s^M_4], WaitPlayed [s^M_5], TrafficInfluenceStop [s^M_6]};
- Act_{Mec} = { palyMessageRequest [t^M_1], Nnef_Traffic_Influence_Create_Res [t^M_2], getMessageRequest [t^M_3], Nnef_EventExposure_Notify_Req(start) [t^M_4], Nnef_EventExposure_Notify_Req(stop) [t^M_5], timerExpire [t^M_6], cancelMessageRequest [t^M_7], Nnef_Traffic_Influence_Delete_Res [t^M_8]};
- $\rightarrow_{Mec}$ = { ($s^M_1\ t^M_1\ s^M_2$), ($s^M_2\ t^M_2\ s^M_3$), ($s^M_3\ t^M_3\ s^M_3$), ($s^M_3\ t^M_4\ s^M_4$), ($s^M_4\ t^M_3\ s^M_4$), ($s^M_4\ t^M_5\ s^M_5$), ($s^M_5\ t^M_3\ s^M_5$), ($s^M_5\ t^M_3\ s^M_6$), ($s^M_6\ t^M_8\ s^M_1$), ($s^M_3\ t^M_7\ s^M_6$), ($s^M_4\ t^M_7\ s^M_7$)}
- s_0^{Mec} = {s^M_1}.

The formal model verification is done by using the concept of bisimulation equivalence [22].

Proposition: T_{app} and T_{Mec} are weakly bisimilar.

Proof: By R_S, it is denoted a relationship between corresponding states of T_{app} and T_{Mec}, such as R_S = {(s^A_1,s^M_1), (s^A_2,s^M_3), (s^A_3,s^M_4), (s^A_4,s^M_5)}. The following transition mapping can be identified:

1. The application initiates a message playing, the media control service invokes traffic influence create procedure and waits for SMF event notifications. The message status becomes pending: For (s^A_1,s^M_1)⊓(s^A_2,s^M_3) ∃ ($s^A_1\ t^A_1\ s^A_2$) ⊓ {($s^M_1\ t^M_1\ s^M_2$), ($s^M_2\ t^M_2\ s^M_3$)}.

2. The application queries the message status while the message is pending: For (s^A_2,s^M_3) ∃ {($s^A_2\ t^A_2\ s^A_2$), ($s^A_2\ t^A_3\ s^A_2$)}⊓ ($s^M_3\ t^M_3\ s^M_3$).

3. The media control service receives a notification on SMF event regarding message playing start and the message status becomes playing. The application queries the message status while the message is playing: For (s^A_3,s^M_4) ∃ {($s^A_2\ t^A_5\ s^A_3$), ($s^A_3\ t^A_3\ s^A_2$)}⊓ {($s^M_3\ t^M_4\ s^M_4$), ($s^M_4\ t^M_3\ s^M_4$)}.

4. The media control service receives a notification on SMF event regarding message playing stop and the message status becomes played. The application queries the message status while the message is played: For (s^A_4,s^M_5) ∃ {($s^A_3\ t^A_6\ s^A_4$), ($s^A_4\ t^A_3\ s^A_4$) }⊓{($s^M_4\ t^M_5\ s^M_5$), ($s^M_5\ t^M_3\ s^M_5$)}.

5. The application considers the message as played and there is no message to be played. The media control service releases the message status and invokes traffic influence delete procedure: For (s^A_4,s^M_5)⊓(s^A_1,s^M_1) ∃ ($s^A_4\ t^A_9\ s^A_1$)⊓{($s^M_5\ t^M_3\ s^M_6$), ($s^M_6\ t^M_8\ s^M_1$)}.

6. The application cancels message playing while the message status is pending and the media control service and invokes traffic influence delete procedure: For (s^A_2,s^M_3)⊓ (s^A_1,s^M_1) ∃{ ($s^A_2\ t^A_7\ s^A_5$), ($s^A_5\ t^A_8\ s^A_1$)}⊓{($s^M_3\ t^M_7\ s^M_6$), ($s^M_6\ t^M_8\ s^M_1$)}.

7. The application cancels message playing while the message status is playing and the media control service and invokes traffic influence delete procedure: For (s^A_3,s^M_4)⊓ (s^A_1,s^M_1) ∃ {($s^A_3\ t^A_7\ s^A_5$), ($s^A_5\ t^A_8\ s^A_1$)}⊓{($s^M_4\ t^M_7\ s^M_6$), ($s^M_6\ t^M_8\ s^M_1$)}.

Therefore, T_{app} and T_{Mec} are weakly bisimilar. ■

Similar models on the media interaction status are maintained by the MEC application and server.

One of the key performance indicators of MEC is latency. The round-trip time (RTT) depends on the time required to transmit packets over the radio interface T_R, the time backhaul connection between the access and core network T_B, the core network processing time T_C, and the time for connection between the core network and MEC server T_T. So, RTT is 2 x (T_R+ T_B+ T_C+T_T.). The proposed mobile edge service assumes MEC server co-location with distributed core functionality, so site the T_T is insignificant. Deployment of distributed core network closer to the edge further reduces the backhaul latency T_B.

For example, Verison engineers have conducted 5G tests with MEC equipment installed at a cell site as a part of cloud radio access network and the tests have shown RTT between 10-15ms [23]. The access to MEC server deployed at a site with distributed core could decrease the latency down to about 15ms which is 25-40% lower than typical 4G deployments [24], [25].

To assess the latency introduces by the proposed API, we emulate the media playing transactions. The configuration experiment setup includes client running on Intel Core i7-3770@3.4 GHz, 8 cores, 8 GB RAM, Ubuntu and server running on Intel Core i7-9750H@2.6 GHz, 6 cores, 16 GB RAM, Ubuntu. The server is implemented using Vert.x, which supports REST-based interface toward the client, and Redis, which writes in Redis store configure to work in single node without clustering [26], [27].

Fig.14 shows the record of operations' latency for a sequence, consisted of 10^5 operations (POST requests and

relevant 201 OK responses). As it might be observed, after the initial system warm-up i.e. after the first $2x10^4$ operations, the operations' latency, seen as a process, converges to about 240µs. The observed spikes in the decimated sequence tend to be between 1 and 15ms and they are caused mainly by the ongoing memory resizing as far as the test operations are exclusively posts to an in-memory repository.

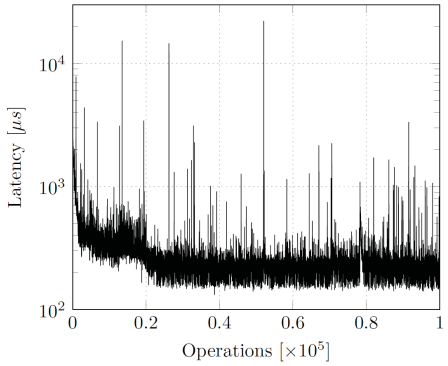

Fig.14 Record of operations' latency for a sequence of 10^5 operations

VI. Conclusion

The session initiation by third party is applicable in many use cases including eHealth, mission critical communications as well as different IoT scenarios. The sessions may be enhanced with message playing or media interactions. In this paper, we study the capabilities to deploy media control for ongoing session with message playing and media interaction at the network edge. These capabilities can be provided by using MEC technology with co-location of the MEC server and distributed core network functions.

We propose REST-based APIs that enable MEC applications to play media session to participant(s) in ongoing session, to play an announcement and collect information from session participant(s), and to record the media interaction. The functionality of the proposed APIs is illustrated by typical use cases. The presented data model describes the RESTful resources and data types. The state of the resources can be manipulated by HTTP methods. Models representing message playing status as seen from MEC application and network point of view are proposed, formally defined, and verified. The formal model verification is useful in assessment of API realization against API specification.

Acknowledgment

This work was supported by grants DH07-10, 2016 and KP-06-H37-33, 2019 from Bulgarian National Science Fund, Ministry of Education and Science.

References

[1] D. Wang, D. Chen, B. Song, N. Guizani, X. Yu and X. Du, "From IoT to 5G I-IoT: The Next Generation IoT-Based Intelligent Algorithms and 5G Technologies," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 114-120, October 2018.

[2] S. Taherizadeh, A. Jones, I. Taylor, Z. Zhao, V. Stankonski, Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review, *Journal of Systems and Software*, vol.136, 2018, pp.19-38.

[3] N. Bazhenov, D. Kozun, Event-Driven Video Services for Monitoring in Edge-Centric Internet of Things Environment, *Proc. of 25th Conference of FRUCT Association, 2019*, pp.47-56.

[4] E. Saksonov, Y. Leokhin, P. Panfilov, Structural Synthesis of the IoT System for the Fog Computing, *Proc. of 24th Conference of FRUCT Association, 2019*, pp.382-387.

[5] L. Zanzi *et al*., "Evolving Multi-Access Edge Computing to Support Enhanced IoT Deployments," *IEEE Communications Standards Magazine*, vol. 3, no. 2, pp. 26-34, June 2019.

[6] Q.V. Pham, et al, "A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art," arXiv:1906.08452v2 [cs.NI] 2 Jan 2020, available at: https://arxiv.org/pdf/1906.08452.pdf

[7] S. Yang, Y. Tseng, C. Huang and W. Lin, "Multi-Access Edge Computing Enhanced Video Streaming: Proof-of-Concept Implementation and Prediction/QoE Models," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1888-1902, Feb. 2019.

[8] S. Wang, "Edge Computing: Applications, State-of-the-Art and Challenges," *Advances in Networks*, vol. 7, Issue 1, 2019, pp. 8-15.

[9] E. Pencheva, I. Atanasov, D. Velkova, V. Trifonov, "Application Level User Traffic Control at the Mobile Network Edge," *Proc. of 24th Conference of FRUCT Association, 2019,* pp.312-327.

[10] 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.

[11] E. Pencheva, I. Atanasov, I, Asenov, "Access and mobility policy control at the network edge, " ISC International Journal of Information Security (ISeCure), August 2019, vol.11, Number 3, pp. 105-111.

[12] 3GPP TS 29.199-11 Technical Specification Group Core Network and Terminals;Open Service Access (OSA); Parlay X Web Services; Part 11: Audio call, Release 9, v9.0.0, 2009.

[13] S. Kekki et al. MEC in 5G Networks, ETSI White Paper No.28, 2018

[14] 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

[15] F. Giust et al. MEC Deployments in 4G and Evolution Towards 5G, ETSI White Paper No.24, 2018

[16] Mukherjee, Shreyasee & Ravindran, Ravi & Raychaudhuri, Dipankar. (2018). A Distributed Core Network Architecture for 5G Systems and Beyond. Proceedings of the Workshop on Networking for Emerging Applications and Technologies, NEAT '18, 2018, pp. 33–38, https://doi.org/10.1145/3229574.3229583

[17] 3GPP TS 29.522 Technical Specification Group Core Network and Terminals;5G System; Network Exposure Function Northbound APIs; Stage 3, Release 15, v15.2.0, 2018

[18] 3GPP TS 23.502 Technical Specification Group Services and System Aspects; Procedures for the 5G System (5GS), Stage 2; Release 16, v16.2.0, 2019

[19] 3GPP TS 29.122 Technical Specification Group Core Network and Terminals; T8 reference point for Northbound APIs, Release 16, v16.3.0, 2019

[20] European Centre for Disease Prevention and Control (ECDC), A literature review on health communication campaign evaluation with regard to the prevention and control of communicable diseases in Europe, Insights into health communication, Technical report, 2019.

[21] K. Rehor, L. Portman, A. Hutton, R. Jain, Use Cases and Requirements for SIP-Based Media Recording, RFC: 6341, 2011.

[22] P. Jančar and S. Schmitz, "Bisimulation Equivalence of First-Order Grammars is ACKERMANN-Complete," *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, Vancouver, BC, Canada, 2019, pp. 1-12.

[23] M. Alleven, Verizon's MEC gear gives an 'edge' in latency, C, https://www.fiercewireless.com/wireless/verizon-s-mec-gear-gives-it-edge-latency

[24] M. Elbamby et al. "Wireless Edge Computing with Latency and Reliability Guarantees," arXiv:1905.05316v1 [cs.NI] 13 May 2019, pp.1-20.

[25] L. Yala, P. A. Frangoudis and A. Ksentini, "Latency and Availability Driven VNF Placement in a MEC-NFV Environment," *2018 IEEE*

Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 2018, pp. 1-7.

[26] Eclipse Vert.x, Available at: https://vertx.io/, Accessed on February 2020.

[27] Redislab 5.0.7, Available at: https://redis.io/, Accessed on February 2020.