Flexible Charging Using Edge Computing

Evelina Pencheva, Ivaylo Atanasov, Denitsa Velkova
Technical University of Sofia
Sofia, Bulgaria
emails: {enp, iia}@tu-sofia.bg, denitsa.velkova@balkantel.net

***Abstract*—Distribution of cloud computing capabilities at the edge of the mobile network addresses the requirements for reduced latency and more efficient usage of network resources. Opening the network edge for third parties enables rapid deployment of innovative applications. In this paper, capabilities for open access to charging function at the mobile edge are studied. New API for online charging is developed which enable MEC system to collaboratively interact with network functions responsible for session management and real time charging. The proposed interfaces follow the adopted Representational State Transfer (REST) used to expose and to access edge cloud services. The API is described by typical use cases, data model and information exchange, and resource structure. Implementation aspects related to modeling the charging application logic and charging status supported by the network are discussed. The latency injected by the proposed API is evaluated theoritically.**

I. Introduction

The fifth generation (5G) networks are designed to address the requirements of large diversity of use cases. The expectations from 5G are to provide enhanced broadband service, ultralow latency, and to support connectivity of huge number of devices in dense environment, which in turn requires revolutionization of charging systems too. To achieve these expectations, 5G design is fundamentally different from its predecessors embedding key technologies such as virtualization, network slicing, and microservices. Virtualization provides cloud environment for both network components and charging systems enabling auto-scaling and fault tolerance. Network slicing allows creation of virtual networks which share the same physical infrastructure and thus supports different use cases opening the door for new charging schemes. Microservices provide self-healing and low latency for network components distributed at the network edge which imposes charging systems to adapt to the same operating modes. In 5G, online charging system and offline charging systems are converged, and the charging interface is a service based on HTTP/JSON [1]. 5G system enables pricing based on peak data rates, rate of user/device mobility, mobile data volume, service deployment time, end-to-end latency, number of connected devices, energy efficiency, etc. [2].

The exponential increase in Internet of Things (IoT) traffic will result in increased number of charging transactions and a massive variety of transactions which requires their offloading and distributed processing. Distribution of charging functionality at the network edge provides scalability, deployment flexibility and evolution. In this paper, the cloud capabilities of the network edge are studied to provide charging services, using Multi-access Edge Computing (MEC) technology.

MEC provides cloud computing and storage capabilities in the vicinity of user's point of attachment to the network [3]. 5G support for MEC includes a significant number of enablers such as network capability exposure, local routing and traffic steering, session and service continuity, quality of service and charging etc. The exposure of policy and charging functionality enables MEC related user traffic to receive the required quality of service treatment and to be charged appropriately [4].

Current research is focused on a new MEC-based service which enables applications to apply flexible online flat-free and volume-based charging schemes. The service provides open access for third party applications to charging functions.

Next sections describe the related works, present the overall service functionality by typical use cases, and specify the service by data model and interface definition. Service state models highlight some implementation issues. The injected latency by the service is evaluated theoretically as a key performance indicator.

II. Related Works

Traditionally, mobile networks adopted different architectures for offline and online charging models, where users chose the payment mode. 5G networks bring more flexibility by combining online and offline systems. 5G supports convergent architecture for both offline and online charging mode, and the choice of online or offline mode depends on the service needs [5], [6]. The 5G charging architecture defines services, operations and procedures of charging using service-based interface specified in [7], [8]. Flexible charging architecture that supports network slicing is presented in [9].

The distributed deployment of core network functions in different edge locations addresses some of the performance issues associated with the centralized control. Due to vicinity to the edge, distributed core architecture reduces latency, saves backhaul resources, improves scalability and supports flexible creation of innovative IoT services. Core network functions can be virtualized and run as virtual machines. In case of distributed core, MEC applications can share the same virtualized infrastructure. The access for MEC applications and services to distributed core functions is provided by Network Exposure Function (NEF) [10]. Regarding charging functionality, NEF provides Nnef_ChargeableParty service which enables management of the chargeable party [11].

MEC adoption sets new business opportunities and challenges for innovative smart charging strategies [12]. Examples include charging mechanisms for virtual resource usage, third party pricing, revenue models for sharing the

physical infrastructure etc. In [13], [14], charging and billing models for MEC applications are described. Different MEC deployment scenarios require specific billing schemes which are presented in [15], [16], [17]. Pricing scheme for MEC-based mobility management enabling flexible charging is proposed in [18]. In [19], the authors describe an architecture enabling MEC billing and charge tracking.

Billing function as a service which is a key feature of MEC is discussed in [20]. There exist also product solutions for billing of IoT services deployed the network edge [21]. MEC-based approach to open access to sponsored data connectivity is presented in [22].

The main contribution of the current research is the extension of capabilities provided to third party applications to manage the chargeable party at the network edge. A MEC-based approach to open access to charging functions is proposed which supports amount and volume reservation, reservation charging and reservation release. Using the proposed API, it is possible to account for each service and to provide accurate billing for multimedia sessions and messaging, as well as to determine the customer's usage. Deployed at the edge, the API features latency reduction required for real-time interactive media. Using the API, analytic applications may allow to deploy more intelligent and flexible charging and billing.

III. Charging as a Service

5G system supports traditional charging models based on session and based on event. Event based charging model is related to user transactions such as message sending, publishing presence information, subscription to presence information, downloading a resource etc., and the service value is preliminary known. Session based charging model is applied when the service value depends on the used volume of network resources (measured in time units or number of sent or received bytes). 5G system supports also new charging models such as flow-based charging and quality of service (QoS) based charging as a part of Policy and Charging Control functionality.

In this paper, a new mobile edge service called Edge-Based Charging (EBC) service is defined. The service enables mobile edge applications to reserve a currency amount from a user account, to add/reduce an amount from existing reservation, to charge a reservation, to release a unused reservation, request conversion of given volume to currency amount, to reserve an amount from user account specified in volume, to reserve additional volume, to charge the reserved volume, to refund the user account by left in the reservation. The proposed service may be beneficial for IoT applications, online gaming, multi-view live streaming of sport events or musical performance, augmented and virtual reality.

An example of an application scenario using the proposed API is a live streaming of World League football match, which is illustrated in Fig.1. A user who wants to watch the football match establishes a secured connection with the service provider supplying his mobile number and other information. Before service provisioning, the user wants to know the cost of the streaming service and the provider's charging application using the EBC interfaces (getAmount) interacts with the rating engine of the mobile operator. The returned charging information may depend on the user location, time of day, user subscription etc., and it is displayed at the user terminal.

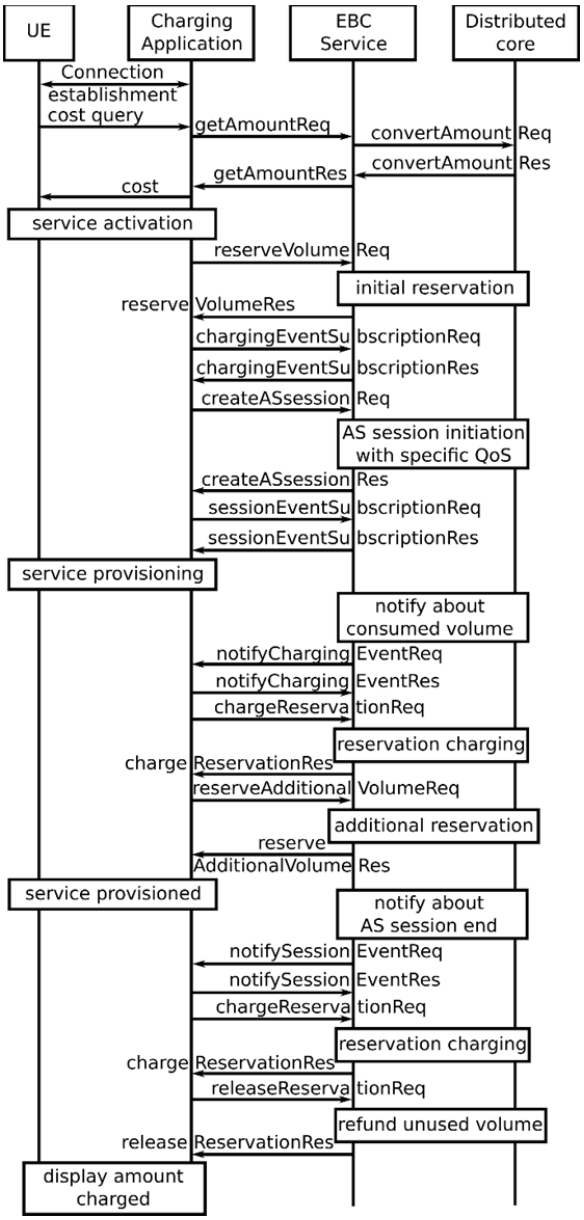

Fig.1 Message flow of advice of charging, volume reservation, charging, additional reservation, and refund of unused reservation

The user activates the streaming service and the service provider's application reserves network resources for the match duration (reserveVolume) to ensure the service accountability. Then the application subscribes with the EBC service to receive notifications related to charging events (subscribeChargingEvents), initiates a session between the user equipment (UE) and the streaming server (create-ASsession),, and subscribes for session related notifications (subscribeSessionEvents). The streaming service starts, and the provider's charging application being notified about charging event (notifyChargingEvent) charges periodically against the reservation (chargeReservation). It happens that the

match must last longer than the allotted time because both teams have an equal score and there is overtime. So, the provider's charging application reserves additional amount (reserveAdditionalVolume) and the user continues watching the match. The match ends with a goal scored by one of the players and the charging application is notified, and then releases the unused amount in the reservation (releaseReservation).

The access of the EBC service to the charging functionality in the operator's network is through the NEF Nnef_ChargeableParty service. The Nnef_ChargeableParty service requests to become a chargeable party for a user session. To initiate a session between the user and Application Service (AS) which provides the service and to provide a specific quality of service (QoS) for the AS session, the EBC service uses the NEF Nnef_AFsessionWithQoS service [11].

IV. Service Resources and Supported Methods

Following RESTful principles for service design, all concepts related to charging are represented as resources located by Uniform Resource Identifiers (URIs).

Fig.2 shows the EBC service resource URIs where the root service URI can be discovered using service discovery register.

The chargingSubscriptions resource and sessionSubscriptions resource represent all subscriptions respectively for charging and session related events. A new {chargingSubscriptionID} resource and a new {sessionSubscriptionID} resource, which represent existing subscription for charging or session events, are created, when the application sends an HTTP POST request to the parent resource. The chargingSubscriptionData data type defines charging event filters, user ID, charging ID, and the address where the application wants to receive notifications. The sessionSubscriptionData data type defines session event filters, user ID, session ID, and the application callback address. The reserveAmounts resource represents all requests for amount reservations. A new {reserveAmountID} resource representing an existing amount reservation request is created when the applications sends an HTTP POST request to the reserveAmounts resource with message body containing user account ID and the information on the charge to be reserved. The 201 Created message contains reservation ID. The chargeReservations resource represents all requests for charging against the reservations and the {chargeReservationID} resource represents an existing request for charging a reservation. When the application creates a new {chargeReservationID} resource, it uses the reservation ID and information on the reservation to be charged. The reserveAdditionalAmounts resource represents all requests for reservation extension and the {reserveAdditionalAmountID} resource represents an existing request for reservation extension. The application uses an HTTP POST request to create a new {reserveAdditionalAmountID} resource providing the reservation ID and information on the charge to be added to the reservation. The releaseReser-vations and {releaseReservationID} resources represent respectively all and an existing request(s) for release of the left reservation. The getAmounts resource serves as all requests for converting given volumes, and the {getAmountID} resource serves as an existing request for converting given volume. An HTTP POST request to the getAmounts resource creates a new request for converting volume and the request body contains ConvertAmountInfo data structure defining the user account to be charged, the volume to be converted and parameters to be used for rating (e.g. "service", "unit", "contact", "operation").

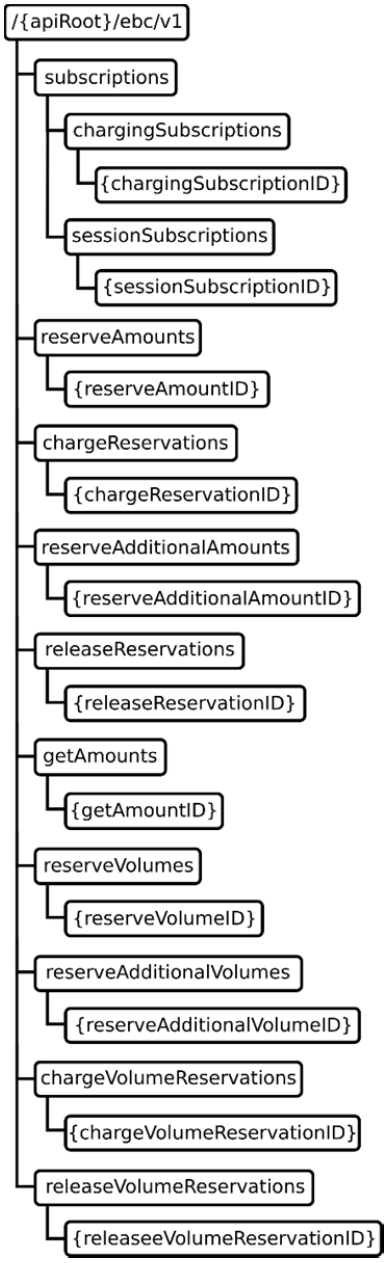

Fig.2 EBC service resources

The reserveVolumes resource, reserveAdditionalVolumes resource, chargeVolumeReservations resource and releaseVolumeReservations resource are related to management of reservations charging by volume and represent all requests respectively for reservations amounts of user accounts, for adding volumes to existing reservations, for charging reservations and for returning of funds left in reservations. Applying an HTTP POST request to the respective resource creates a new child resource. The reserveVolumeData data type defines information required to

create a new reserveVolumeID resource and contains the user account, which is subject to reservation, the volume of reservation, the description information to appear on the bill and parameters to perform rating. The reserveAddVolumeData type defines information required to add a volume to an existing reservation and contains reservations identifier, the volume to be added to the reservations and billing text. The chargeReservationData data type is used to charge a reservation and contains reservation identifier, the volume to be charged, billing text and reference code which uniquely identifies the request in case of disputes. The release-ReservationData data type points the reservations identifier and it is used to return funds left in a reservation.

All resources support also HTTP GET method which retrieves information about the respective resource. The leave resources support HTTP PUT method which updates an existing resource and HTTP DELETE method which deletes an existing resource.

V. Charging Models

Modeling the state of a charging transaction is a part of EBC service implementation.

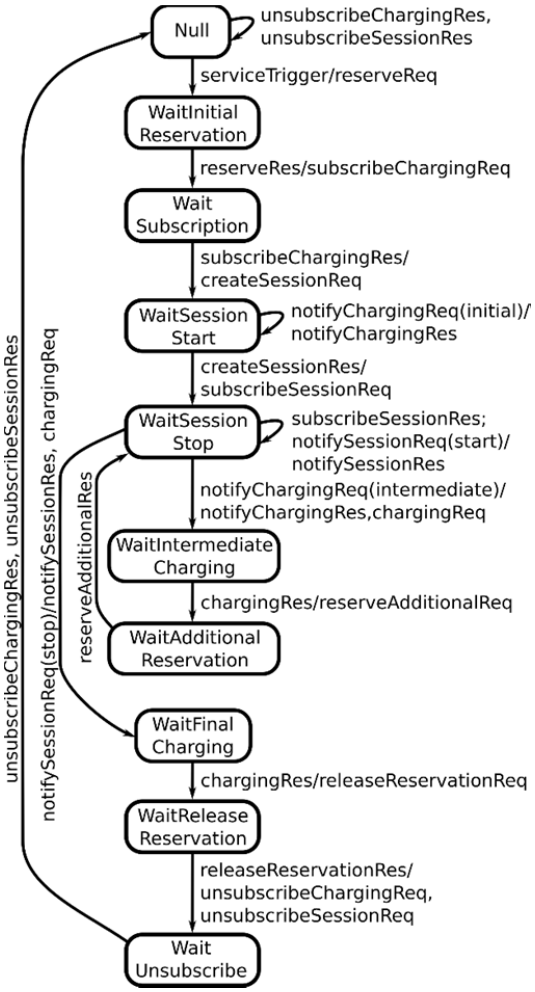

Fig.3 Charging state model, supported by a mobile edge charging application

Fig.3 shows the simplified model representing the state of charging from the application point of view. On service trigger, the application makes initial reservation and subscribes to receive charging related notifications. In case of successful reservation, the application starts a session and subscribes for notifications about session related events. When the reservation is consumed, the application requests charging the reservation and makes additional reservation. The application is notified about session stop, it requests charging for the consumed resources and releases the left in the reservation.

Fig.4 shows a simplified charging state model supported by the MEC platform. To make reservation in the network, the EBC service invokes Nnef_ChargeableParty_Create operation of NEF. Notifications about charging related events are provided to the EBC service by Nnef_ChargeableParty_Notify operation. To charge the reservation and to make additional reservation the EBC service invokes Nnef_ChargeableParty_Update operation. Following the application instructions, the EBC service invokes Nnef_AFsessionWithQoS_Create operation to create an Application Server session and to provide specific QoS. The EBC service is notified about session related events by Nnef_AFsessionWithQoS_Notify operation.

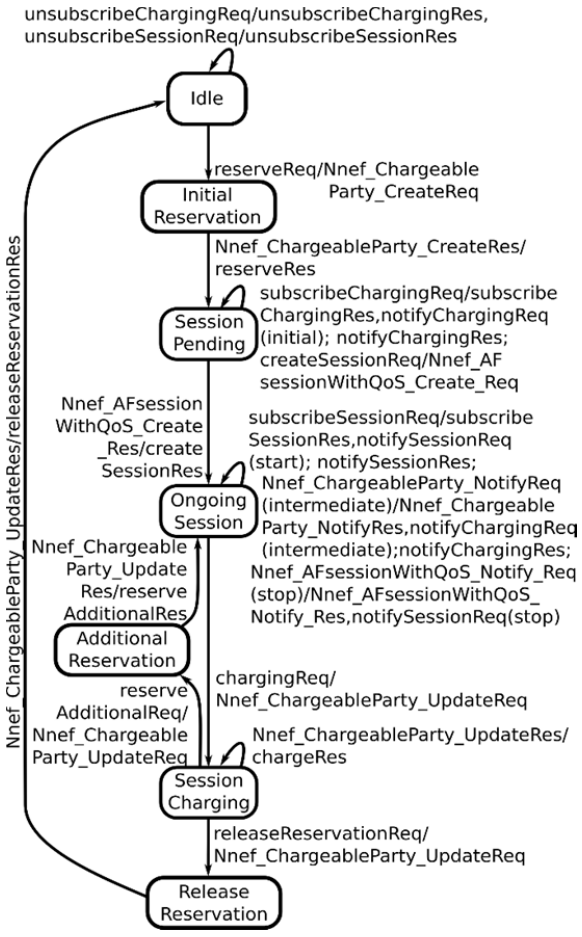

Fig.4 Charging state model, supported by a mobile edge platform

Model verification is done by formal model definition using the concept of finite state automata (FSA) and the concept of weak bi-simulation [23]. An FSA is defined as a quadruple of a set of states, a set of labels, a state of labeled transitions, and a set of initial states. The concept of bi-simulation is used in parallel programming to formalize the idea of behavioral equivalence of processes. In the following formal definitions, short names are given in brackets.

By $F_{App} = (S_{App}, Act_{App}, \rightarrow_{App}, s_0^{App})$ it is denoted the FSA representing the charging state model supported by a mobile edge application, where:

- S_{App} = {Null [s^A_1], WaitInitialReservation [s^A_2], WaitSubscription [s^A_3], WaitSessionStart [s^A_4], WaitSessionStop [s^A_5], WaitIntermediateCharging [s^A_6], WaitAdditionalReservation [s^A_7], WaitFinalCharging [s^A_8], WaitReleaseReservation [s^A_9], WaitUnsubscrbe [s^A_{10}]};
- Act_{App} = {serviceTrigger [t^A_1], reserveRes [t^A_2], subscribeChargingRes [t^A_3], notifyChargingReq(initial) [t^A_4], createSessionRes [t^A_5], subscribeSessionRes [t^A_6], notifySessionReq(start) [t^A_7], notifyChargingReq(intermediate) [t^A_8], chargingRes[t^A_9], reserveAdditionalRes [t^A_{10}], notifySessionReq(stop) [t^A_{11}], releaseReservationRes [t^A_{12}], unsubscribeChargingRes [t^A_{13}], unsubscribeSessionRes [t^A_{14}]};
- $\rightarrow_{App}$ = {(s^A_1 t^A_1 s^A_2), (s^A_2 t^A_2 s^A_3), (s^A_3 t^A_3 s^A_4), (s^A_4 t^A_4 s^A_4), (s^A_4 t^A_5 s^A_5), (s^A_5 t^A_6 s^A_5), (s^A_5 t^A_7 s^A_5), (s^A_5 t^A_8 s^A_6), (s^A_6 t^A_9 s^A_7), (s^A_7 t^A_{10} s^A_5), (s^A_5 t^A_{11} s^A_8), (s^A_8 t^A_9 s^A_9), (s^A_9 t^A_{12} s^A_{10}), (s^A_{10} t^A_{13} s^A_1), (s^A_{10} t^A_{14} s^A_1), (s^A_1 t^A_{13} s^A_1), (s^A_1 t^A_{14} s^A_1)};
- s_0^{App} = {s^A_1}.

By F_{Mec} it is denoted the charging state model supported by the MEC platform $T_{Mec} = (S_{Mec}, Act_{Mec}, \rightarrow_{Mec}, s_0^{Mec})$ where:

- S_{Mec} = {Idle [s^M_1], InitialReservation [s^M_2], SessionPending [s^M_3], OngoingSession [s^M_4], SessionCharging [s^M_5], AdditionalReservation [s^M_6], AdditionalReservation [s^M_6], ReleaseReservation [s^M_7]};
- Act_{Mec} = { reserveReq [t^M_1], Nnef_ChargeableParty_CreateRes [t^M_2], subscribeChargingReq [t^M_3], notifyChargingReq [t^M_4], createSessionReq [t^M_5], Nnef_AFsessionWithQoS_Create_Res [t^M_6], subscribeSessionReq [t^M_7], notifySessionRes [t^M_8], Nnef_ChargeableParty_NotifyReq (intermediate) [t^M_9], Nnef_AFsessionWithQoS_Notify_Req(stop) [t^M_{10}], chargingReq [t^M_{11}], Nnef_ChargeableParty_UpdateRes [t^M_{12}], reserveAdditionalReq [t^M_{13}], releaseReservationReq [t^M_{14}], unsubscribeChargingReq [t^M_{15}], unsubscribeSessionReq [t^M_{16}]};
- $\rightarrow_{Mec}$ = { (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_3), (s^M_3 t^M_5 s^M_3), (s^M_3 t^M_6 s^M_4), (s^M_4 t^M_7 s^M_4), (s^M_4 t^M_8 s^M_4), (s^M_4 t^M_9 s^M_4), (s^M_4 t^M_9 s^M_4), (s^M_4 t^M_4 s^M_4), (s^M_4 t^M_{10} s^M_4), (s^M_4 t^M_{11} s^M_5), (s^M_5 t^M_{12} s^M_5), (s^M_5 t^M_{13} s^M_6), (s^M_6 t^M_{12} s^M_4), (s^M_5 t^M_{14} s^M_7), (s^M_7 t^M_{12} s^M_1), (s^M_1 t^M_{15} s^M_1), (s^M_1 t^M_{16} s^M_1)}
- s_0^{Mec}= {s^M_1}.

Strong bi-simulation requires strict matching of transitions in both FSAs. In weak bi-simulation, invisible (internal) transitions may be disregarded.

Proposition: F_{App} and F_{Mec} are weakly bi-similar i.e. they expose equivalent behavior.

Proof: Let $R \subseteq (S_{App} \times S_{Mec})$ where R = {(s_1^A, s_1^M), (s_2^A, s_2^M), (s_5^A, s_4^M), (s_6^A, s_6^M), (s_7^A, s_6^M), (s_9^A, s_7^M)}. The states in R reflect no charging and no session, initial reservation for the session, ongoing session, charging the session, additional reservation for the session, and release reservation for the ended session. Then the following matching between transitions of F_{App} and F_{Mec} can be identified:

1. On service trigger, the application makes initial reservation, subscribes for charging events, receives a notification about initial charging status, and creates a session between the user and application server: for {(s^A_1 t^A_1 s^A_2) ⊓ (s^A_2 t^A_2 s^A_3) ⊓ (s^A_3 t^A_3 s^A_4) ⊓ (s^A_4 t^A_4 s^A_4) ⊓ (s^A_4 t^A_5 s^A_5)} ∃ (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_3) ⊓ (s^M_3 t^M_5 s^M_3) ⊓ (s^M_3 t^M_6 s^M_4)}.
2. The application subscribes for session events and receives notifications: for {(s^A_5 t^A_6 s^A_5) ⊓ (s^A_5 t^A_7 s^A_5)} ∃{(s^M_4 t^M_7 s^M_4) ⊓ (s^M_4 t^M_8 s^M_4)}.
3. During the session, the granted resource are consumed, the application is notified, and it requests intermediate charging: for (s^A_5 t^A_8 s^A_6) ∃ {(s^M_4 t^M_9 s^M_4) ⊓ (s^M_4 t^M_4 s^M_4) ⊓ (s^M_4 t^M_{11} s^M_5) ⊓ (s^M_5 t^M_{12} s^M_5)}.
4. The application requests additional reservation: for {(s^A_6 t^A_9 s^A_7) ⊓ (s^A_7 t^A_{10} s^A_5)} ∃ {(s^M_5 t^M_{13} s^M_6) ⊓ (s^M_6 t^M_{12} s^M_4)}.
5. The session ends in the network, the application is notified, requests final charging and refund of unused reservation: for {(s^A_5 t^A_{11} s^A_8) ⊓ (s^A_8 t^A_9 s^A_9)} ∃ {(s^M_4 t^M_{10} s^M_4) ⊓ (s^M_4 t^M_{11} s^M_5) ⊓ (s^M_5 t^M_{14} s^M_7)}.
6. The application terminates its subscription for charging events and session events: for {(s^A_9 t^A_{12} s^A_{10}) ⊓ (s^A_{10} t^A_{13} s^A_1) ⊓ (s^A_{10} t^A_{14} s^A_1) ⊓ (s^A_1 t^A_{13} s^A_1) ⊓ (s^A_1 t^A_{14} s^A_1)} ∃ (s^M_7 t^M_{12} s^M_1) ⊓{(s^M_1 t^M_{15} s^M_1) ⊔ (s^M_1 t^M_{16} s^M_1)}.

Therefore, F_{App} and F_{Mec} are weakly bi-similar. ■

The concept of bi-similarity is useful at the phase of software verification to test the process parallel execution and at the phase of system validation to prove compliance of software specification with its implementation.

VI. API Performance Evaluation

It is widely recognized that the main MEC Key Performance Indicators (KPI) for service providers and mobile operators are delay improvement thanks to the proximity of users, network performance and cost saving due to more efficient usage of transport and backhaul network, energy efficiency due to usage of small servers, and more efficient management of computation and networking resources due to network function virtualization [24].

In [25], MEC performance metrics are described, which demonstrates the technology benefits. MEC metrics are as functional and non-functional ones. Functional MEC metrics impact on user perception and include latency, energy efficiency, throughput, loss rate, jitter etc. Non-functional MEC

metrics depend on deployment and management scenarios and consider service lifecycle, service fault tolerance and availability, processing load etc.

In this Section, the focus is on latency injected by the proposed API. It is evaluated theoretically so called Service Delivery Time (SDT). With the proposed EBC service, the charging application is triggered on notification about chargeable event occurred in the network and it sends instructions for charging. So, the latency injected by the EBC service is the time which is taken to a network request (e.g. notification about session start) to reach the application, being processed and transferred back to the network (e.g. charging instructions sent by the application). SDT is a sum of Service Processing Time (SPT) and Round-Trip Time (RTT).

SPT is a non-functional metric representing the time to process the request and it depends on computational load. RTT is a functional metric and it is the time taken for a request, generated from the network to go to the charging application be replied and travel back.

Fig.5 shows the sequence diagram illustrating the message exchange used to evaluate the EBC latency. Upon occurrence of chargeable event, the network notifies the EBC service and it is the beginning of SDT period.

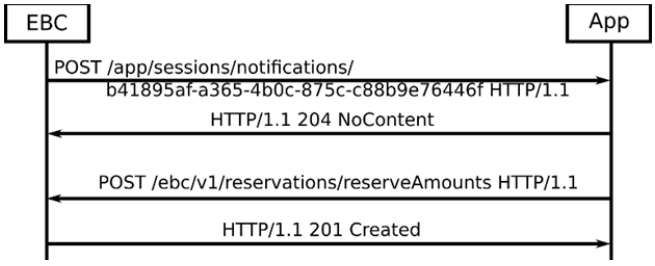

Fig.5 Typical message exchange in the context of EBC service

The EBC service sends a POST request to the application's callback address to notify the application about the chargeable event. The time budget required for notification is a sum of the time required by the EBC service to create the notification request (T_{notify}^{n}), the time for norification request transfer, and the time for notification request processing by the application (T_{notify}^{a}), plus the time required by the application to create the notification response (T_{204}^{a}), the time for notification response transfer, and the time for notification response processing by the EBC service (T_{204}^{a}). The time budget required for sending charging instructions is a sum of the time required by the application to create the charging request (T_{charge}^{a}), the time for charging request transfer, and the time for charging request processing by the EBC service (T_{charge}^{n}), plus the time required by the EBC service to create the charging response (T_{200}^{n}), the time for charging response transfer, and the time for charging response processing by the application (T_{200}^{a}). As the mobile edge application and the EBC service run on the same virtualized platform, the message transfer time is negligible. So, the SDT may be theoretically evaluated as:

$$SDT=T_{notify}^{n}+T_{notify}^{a}+T_{204}^{a}+T_{204}^{n}+T_{charge}^{a}+ +T_{charge}^{n}+T_{200}^{n}+T_{200}^{a}. \quad (1)$$

An example of a POST request for notification about chargeable event, where the JSON body of the request contains information about the time stamp, session ID, user ID, and event type, looks like the following:

```
POST /app/sessions/notifications/70107427-c772-4122-a1a8-
e69abe63ca10 HTTP/1.1
host: [::1]:8080
user-agent: Vert.x-WebClient/3.9.1
content-type:application/json
accept:application/json
content-length: 149

{"timeStamp":"2020-07-10T13:50:11.930Z","session":"c17d668e-
2eca-4387-9c82-5886a764a836","eventType":"sessionStart",
"userID":"ac361f9c@example.com"}
```

The respective response of the POST request for notification about chargeable event looks like the following:

```
HTTP/1.1 204 No Content
Location: 70107427-c772-4122-a1a8-e69abe63ca10
```

An example of a POST request with charging instructions, where the JSON body of the request contains information about the session ID, the user account ID, a reference code (in case of disputes), the volume, and the units, and the respective response look like the following:

```
POST /ebc/v1/reserveAmounts HTTP/1.1
user-agent: Vert.x-WebClient/3.9.1
content-type: application/json
accept: application/json
content-length: 189
host: ::1:8088

{"session":"70107427-c772-4122-a1a8-
e69abe63ca10","userAccountID":"7ca87145-c349-480f-ab7c-
e0adf0f2c7ef","units":"minute","volume":20,"referenceCode":"54fd
e971-30a7-4d91-8152-7566623c02ec"}


HTTP/1.1 201 Created
location: http://::1:8088/ebc/v1/reserveAmounts/e5bb38b0-9adc-
4b9c-a659-0daef7b443d9
content-type: application/json
content-length: 246

{"session":"70107427-c772-4122-a1a8-
e69abe63ca10","userAccountID":"7ca87145-c349-480f-ab7c-
e0adf0f2c7ef","units":"minute","volume":20,"referenceCode":"54fd
e971-30a7-4d91-8152-
7566623c02ec","reserveAmountID":"e5bb38b0-9adc-4b9c-a659-
0daef7b443d9"}
```

We use this typical message exchange in the context of the proposed EBC service to evaluate theoretically the injected latency. The message creation/processing time is a product of input message size in bits (L), the complexity of the message as necessary CPU cycles per bit (X) over the MEC host's CPU frequency (f). Let, the MEC host's CPU clock speed is of 3.2 GHz and the complexity is 1000 cycles per bit [26]. As the charging application and EBC service are executed on the same MEC host, the time budget required to create a message and the time budget required to process the same message may be regarded as equals. So,

$$SDT= 2T_{notify}+2\ T_{204}+2T_{charge}+2T_{200}. \quad (2)$$

$$T_{notify}=L_{Notify}.X/f = 0.87\ ms \quad (3)$$

$$T_{204}=L_{204}.X/f = 0.003\ ms \quad (4)$$

$$T_{charge}=L_{charge}.X/f = 0.87\ ms \quad (5)$$

$$T_{201}=L_{201}.X/f = 1\ ms \quad (6)$$

So, the injected latency for sending a notification about a chargeable event to a mobile edge application and receiving charging instruction by the network is theoretically evaluated as SDT= 5.5 *ms*.

VII. Conclusion

The variety of use cases supported by 5G system requires development of flexible charging systems too. The open access to charging functionality creates new opportunities to application and content providers which can play profitable and complementary roles. The edge computing which ensures more efficient network operation, better service delivery and personal user experience may contribute to better monetarization of mobile broadband experience. Delegating the charging functionality to mobile edge applications enables increase of the edge responsiveness and thus proactive user experience maintenance.

The paper presents an approach to open access to charging functions at the edge of the mobile network. Using the proposed mobile edge interfaces, charging applications may reserve amounts or volumes, may charge reservation, may extend reservation, and may refund unused reservations. Future work will be aimed at extending the proposed functionality with capabilities of direct charging by volume and amount as well as to emulate the proposed functionality.

References

[1] 3GPP TS 32.240, Telecommunication management; Charging management, Charging architecture and principles, Release 16, 2020.

[2] F. L. Rodriges, U. S. Dias, D. R. Campelo, R. O. Albuquerque, A. J. Lim, L. J. G. Villalba. "QoS management and Flexible Traffic Detection Architecture for 5G Mobile Networks," *Sensors* (Basel) 2019 vol. 19, issue 6,.doi: 10.3390/s19061335

[3] ETSI GS MEC 002 Multi-access Edge Computing (MEC); Phase 2: use cases and Requirements, v2.1.1, 2018.

[4] Atanasov, I., E. Pencheva, A. Nametkov, V. Trifonov. "On Functionality of Policy Control at the Network Edge," *International Journal on Information Technologies and Security*, No. 3 (vol.11), 2019, pp. 3-24.

[5] S. Velrajan. "5G pricing – How would Service Providers monetize 5G investments?," 2019, Available at: https://www.thetech.in/2019/04/5g-pricing-how-would-service-providers.html

[6] R. Tornkvist, C. Shan. "Charging and Billing Architecture for 5G networks, *Journal of ICT Standardization*, vol.7, issue 2, 2019, pp185-194.

[7] 3GPP TS 32.290 Telecommunication management; Charging management, 5G system;Services, operations and procedures of charging using Service Based Interface (SBI), Release 16, 2020.

[8] L. Bonati, M. Polese, S. D'Oro, S. Basagni, T. Melodia. "Open, Programmable, and Virtualized 5G networks: State-of-the-Art abd the Road Ahead, ", *Computer Science*, arXiv:2005.10027v2 [cs.NI] 21 May 2020, pp. 1-66.

[9] A. Barakabitze, A. Ahmad, R. Mijumbi, A. Hines. "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol.167, 2020, pp.1-40.

[10] 3GPP TS 23.502 Procedures for 5G System, Stage 2, Release 16, 2020.

[11] 3GPP TS 29.522 5G System; Network Exposure Function Northbound APIs; Stage 3, Release 15, v15.2.0, 2018.

[12] J. Zhang, Z. Wu, W. Xie and F. Yang, "MEC Architectures in 4G and 5G Mobile Networks," *10th International Conference on Wireless Communications and Signal Processing (WCSP)*, Hangzhou, 2018, pp. 1-5, doi: 10.1109/WCSP.2018.8555652.

[13] Q. V. Pham et al, "A Survey of Multi-Access Edge Computing in 5G and beyond: Fundamentals, Technology Integration, and State-of-the-Art," *IEEE Communications Surveys and Tutorials*, 2020, arXiv:1906.08452, pp.1-43.

[14] Y. Li, K.H. Kim, C. Vlachou, J. Xie, "Bridging the Data Charging Gap in the Cellular Edge, " *Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '19), ACM*, New York, NY, USA, 2019, pp.1-14,, doi:10. 1145/3341302.3342074

[15] Z. Wang and Y. Cai, "Management Optimization of Mobile Edge Computing (MEC) in 5G Networks," *IEEE International Conference on Communications Workshops (ICC Workshops*), Shanghai, China, 2019, pp. 1-6, doi: 10.1109/ICCW.2019.8756650.

[16] D. Sabella, A. Reznik, R. Frazao, "Multi-Access Edge Computing in Action, " *CRC Press,* 2019.

[17] "China Unicom Edge Computing Technology, " Chine Unicom, Whitepaper, 2017.

[18] D. Sabella, N. Smith, N. Oliver, K. A. Doshi, S. Prabhakaran, M. Filippou, F. Guim Bernat, "Multi-access Edge Computing (MEC) Billing and Charging Tracking Enhancements, " United States Patent Application 20190158300, 2018, Available at: http://www.freepatent sonline. com/y2019/0158300.html.

[19] J. Lee, D. Kim, J. Lee, "ZONE-Based Multi-Access Edge Computing Scheme for User Device Mobility management," *Applied Sciences*, vol.9, issue 11, 2019, pp.1-16, doi:10.3390/app9112308.

[20] P. Varga, et al. "5G Support for Industrial IoT Applications – Challenges, Solutions, and Research Gaps," *Sensors* 2020, 20, 828; doi:10.3390/s20030828, pp.1-43.

[21] R. Zhang, "A Convergent Billing System for the 5G Era, " pp.22-24, *Intelligence Empowers Your Business Success*, Huawei, Mobile World live, issue 2, February 2020.

[22] I. Atanasov, E. Pencheva, I. Asenov, V. Trifonov, "Sponsored Data Connectivity at the Network Edge, " *4th International Conference on Advanced Computing and Data Sciences*, La Valetta, Malta, 2020, pp.1-9.

[23] A. Halchin, Y. Ait-Ameur, N. K. Singh, A. Feliachi and J. Ordioni, "Certified Embedding of B Models in an Integrated Verification Framework," *International Symposium on Theoretical Aspects of Software Engineering (TASE)*, Guilin, China, 2019, pp. 168-175.

[24] I. Hussain, Q. Duan, T. Zhong. "Service performance tests on the on the Mobile Edge Computing Platform: Challenges and Opprtunities, " in Edited book "*Smart Service Systems, Operations, Management, and Analytics*, " Springer, 2019, pp.1-7.

[25] ETSI GS MEC-IEG 006; "Mobile Edge Computing; Market Acceleration; MEC Metrics Best Practice and Guidelines, " v1.1.1, 2017

[26] K. Cheng, Y. Teng, W. Sun, A. Liu and X. Wang, "Energy-Efficient Joint Offloading and Wireless Resource Allocation Strategy in Multi-MEC Server Systems," in *Proc. of IEEE International Conference on Communications (ICC),* Kansas City, MO, 2018, pp. 1-6.