Mobile Edge Service for Charging Control

Ivaylo Atanasov, Evelina Pencheva, Aleksandar Nametkov, Ventsislav Trifonov
Technical University of Sofia
Sofia, Bulgaria
{iia,enp}@tu-sofia.bg, aleksandar.nametkov@balkantel.net, vgt@tu-sofia.bg

***Abstract*—5G systems will provide a flexible charging capabilities including policy control based on spending limits for a subscription. Edge computing possesses the potential for exposing policy control and charging capabilities to third party applications close to the end users. In this paper, we propose a new mobile edge service that provides open access to functions related to monitoring the usage limits regardless the way of measurement i.e. monetary, volume, duration, etc. Currently, this function is a part of the core network functionality. Using the proposed service interfaces, an intended application may track the usage of network resource by a subscriber and to request applying of enforcement action to user traffic such as quality of service downgrade, traffic redirection or blocking close to the end user. The description of the proposed mobile edge service is provided in form of typical use cases, data model, interface definition, and formally verified state models.**

I. Introduction

Future fifth generation (5G) networks are coming with the promise for improved network performance at competitive costs and enhanced functionality in order to support innovative services. 5G is expected to improve significantly the customer quality of experience in the context of growing data traffic, requirements of higher bandwidth, and low latency [1], [2], [3], [4].

One of the key technologies for enhancing network functional and architectural viability, including increased autonomy and reduced capital expenditure, is Network Function Virtualization (NFV) [5]. The NFV can address the 5G design challenges through virtualized network resources, computing and storage functionality, and service abstraction [6], [7].

The conceptual architectural framework of 5G includes both traditional cloud deployments and edge deployments [8]. Multi-access Edge Computing (MEC) is an attempt to improve modularity and scalability of the network by disaggregation of cloud capabilities in the vicinity to end users. While NFV provides flexibility by dynamic network function deployment, MEC brings intelligence at the network edge [9], [10], [11].

The mobile edge Radio Network Information Service (RNIS) provides real-time information about radio network conditions to applications [12]. Typical information that may be provided includes up-to-date radio network information regarding radio network conditions; measurement information related to the user plane based on 3GPP specifications; information and changes in information about user equipments (UEs) connected to the radio node(s) associated with the mobile edge host and related radio access bearers.

The mobile edge UE Identity Service may be used by applications to register a tag, representing UE [13]. The purpose of this service is to enable applying traffic rules for specific UE.

As to MEC technical requirements the mobile edge system shall allow the collection of charging-related information, log it in a secure way and make it available for further processing [14]. Charging-related information can include traffic usage, application instantiation, access, usage duration, resource usage etc.

In this paper, we propose a new mobile edge service named Charging Control Service (CCS) which might enable open access to charging related information. The proposed service exposes functionality for track spending applicable to a subscriber.

The rest of the paper is organized as follows. Next section presents the research motivation, discussing the benefits of distributing subscriber spending limits functionality at the network edge. Section III describes the extended functionality. Section IV presents the data model and data types which are used by service interfaces, and Section V presents the resource structure and methods supported by the resources. In section VI, some implementation details are discussed, including models representing the mobile edge application logic and the resource states as seen by the networks. Models are formally described and it is proved in a mathematical way that they expose equivalent behavior.

II. Research Motivation

Policy and Charging Control is an important component of 5G core networks that brings together and enhances capabilities from previous generations to deliver dynamic control of policy and charging on a per subscriber and per IP flow basis [15]. It encompasses Flow Based Charging for network usage, including charging control and online credit control, for service data flows and application traffic and Policy control for session management and service data flows (e.g. gating control, quality of service (QoS) control, etc.).

Subscriber spending limits is a function that enables policy decisions based on the status of policy counters that are maintained in the Online Charging System (OCS) [16]. Policy counter is a mechanism to track spending applicable to a subscriber. The policy counter status is a label whose values are not standardized and that is associated with a policy counter's value relative to the spending limit(s). The number of possible policy counter status values for a policy counter is one greater than the number of thresholds associated with that policy counter, i.e. policy counter status values describe the status around the thresholds. This is used to convey information relating to subscriber spending from OCS [17].

We propose to relocate the subscriber spending limit function from the core network to the network edge. This will enable more timely reaction in case of reaching a threshold defined for a policy counter, i.e. policy-based decisions may be enforced close to the end user.

Some of the applications related to control on spending limits are as follows.

The operator and the customer may negotiate spending limits (e.g. volume, duration monetary) and after which the customer traffic will be automatically cut-off. Upon approaching the negotiated thresholds a warning message may be sent to the customer. Similar use case enables an agreement for higher data speeds for predefined period of time or data volume after which the user traffic is shaped.

Another use case is when the customer has a prepaid data service, which allows consumption of a specified data volume. When this volume is reached, the HTTP traffic is redirected to specific application server to enable recharging in order to purchase a supplementary volume of data that may be consumed.

Fig.1 shows the proposed architecture for deployment of policy decisions based on subscriber spending limits.

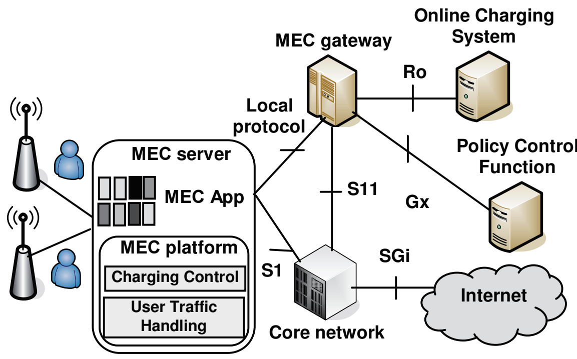

Fig.1 Architecture of MEC system for policy decisions based on subscriber spending limits

The MEC platform is located at the network edge between the bases station / aggregation point / access point and the Evolved Packet Core. It may be bundled in or located in proximity of the radio node. The MEC platform provides mobile edge services such as services for charging control and user traffic handling. The MEC server hosts the mobile edge applications and the MEC platform. The MEC platform exposes the proposed CCS and Bandwidth Management Service (BWMS). The BWMS allow different mobile edge applications to request specific bandwidth requirements (bandwidth size, bandwidth priority, or both). The BWMS may aggregate all the requests and act in a manner that will help to optimize the bandwidth usage [18]. The MEC server may generate or manipulate user traffic. As some of the user traffic may not pass through the core network, the MEC gateway may perform charging and eventually lawful interception. In addition to minimization of latency, it is possible to steer the user traffic on a per session/packets bases. The OCS provides real time charging related information. The Policy Control Function provides policy rules to control plane functions to enforce them.

An example use case for subscriber spending limits is illustrated in Fig.2. The example shows a scenario where a user starts watching a video on demand application. The mobile edge application registers with the BWMS the video bandwidth requirements for premium rate and the user enjoys the high quality video experience. When the user reaches a certain spending limit, the mobile edge application decides to downgrade the rate for the video stream and it deregisters the video application for premium bandwidth allocation.

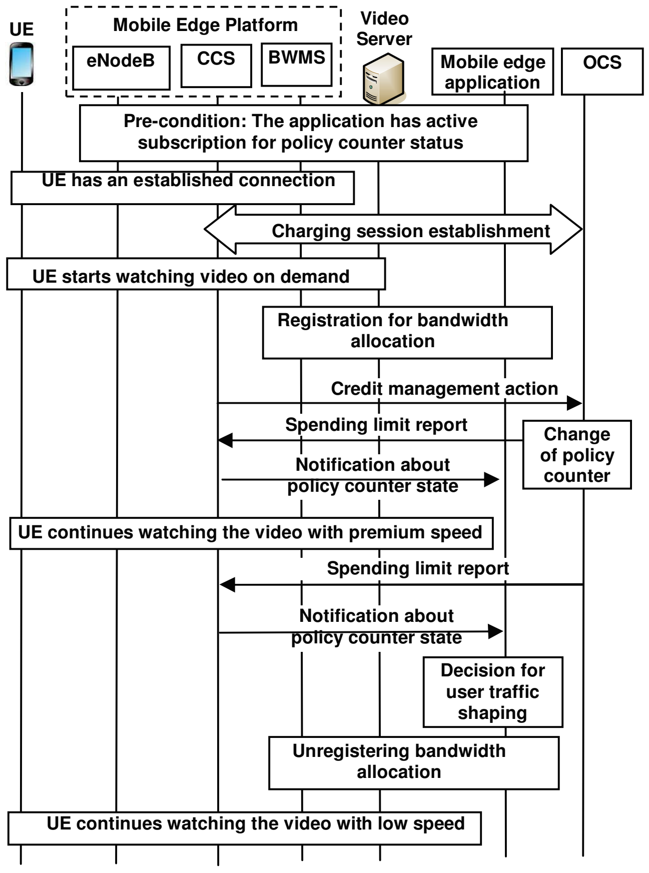
UE
Mobile Edge Platform
eNodeB
CCS
BWMS
Video Server
Mobile edge application
OCS
Pre-condition: The application has active subscription for policy counter status
UE has an established connection
Charging session establishment
UE starts watching video on demand
Registration for bandwidth allocation
Credit management action
Spending limit report
Change of policy counter
Notification about policy counter state
UE continues watching the video with premium speed
Spending limit report
Notification about policy counter state
Decision for user traffic shaping
Unregistering bandwidth allocation
UE continues watching the video with low speed

Fig.2 Use case of policy based-decision on spending subscriber limits

Generally, the ability to monitor the subscriber spending limits is the foundation of many mobile charging plans. Distributing the policy-based functionality based on tracking the subscription spending at the network edge provides more flexibility in packet filter handling and application level charging.

III. DESCRIPTION OF SERVICE FUNCTIONALITY

The proposed mobile edge Charging Control Service (CCS) provides access to the status of policy counter(s) related for a user through:

- Request for the policy counter(s) status of a subscriber;

- Notification upon change in policy counter(s) status.

The communication between the mobile edge services and applications follows the Representational State Transfer (REST) style.

Fig.3 shows the message flow for application requesting policy counter information. For a mobile edge application to determine the status of policy counter, it sends a GET request to the CCS providing subscriber's and policy counter's identification. The MEC platform opens a dialogue with the OCS to retrieve information about the policy counter requested. The CCS receives a response with the status of the policy counter requested.

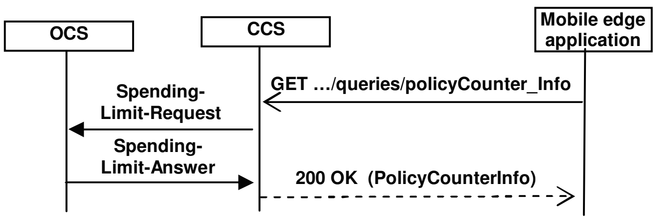

Fig.3 Flow of application requesting policy counter information

To receive notification about policy counter change, the mobile edge application creates a subscription to the related event that is available at CCS. Fig.4 shows a scenario where a mobile edge application sends POST request to the CCS to create a subscription for notifications about policy counter change. The request body contains **PolicyCounterData** data structure to the resource, representing the subscription. The application includes in **PolicyCounterData** structure the address where it wishes to receive notifications. The CCS returns a response with message body containing **PolicyCounterData** data structure. The data structure contains the address of the resource creates and the subscribed event type.

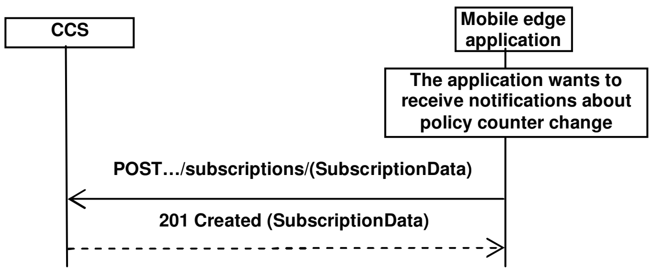
CCS
Mobile edge application
The application wants to receive notifications about policy counter change
POST.../subscriptions/(SubscriptionData)
201 Created (SubscriptionData)

Fig.4 Flow of subscribing to the policy counter change information

CCS may define an expiry time for the subscription to policy counter events. In case expiry time is used, the time is included in the **PolicyCounterData** structure in the response message to the subscription request. On subscription expiry, CCS sends a POST request with a notification to the callback address provided by the application that owns the subscription, as shown in Fig.5.

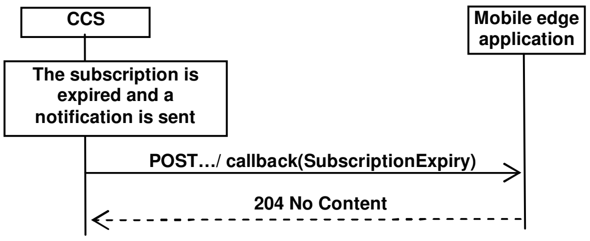
CCS
Mobile edge application
The subscription is expired and a notification is sent
POST.../ callback(SubscriptionExpiry)
204 No Content

Fig.5 Flow of CCS sending notification on subscription expiry

In case of subscription expiry, the mobile edge application needs to update the subscription for policy counter status events. It sends a PUT request to the resource representing the subscription, as shown in Fig.6. The message body of the response contains the accepted data structure specific to that subscription.

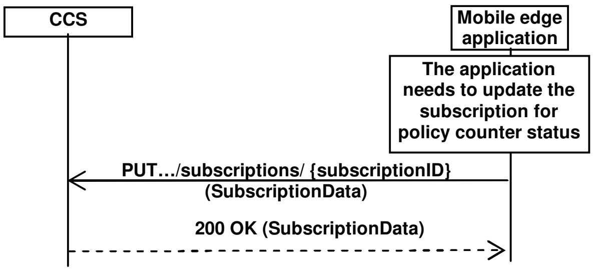
CCS
Mobile edge application
The application needs to update the subscription for policy counter status
PUT.../subscriptions/ {subscriptionID} (SubscriptionData)
200 OK (SubscriptionData)

Fig.6 Flow of subscription modification

Following the same message pattern the mobile edge application may terminate the subscription to policy counter status events. It sends a DELETE request to the resource representing the respective subscription.

A mobile edge application can be notified of the status of policy counter in case of status change. Fig.7 shows the message flow for notification about policy counter status change.

When the OCS detects a status change of the policy counter of interest it reports the change to the mobile edge platform. The CCS sends a POST request with message body containing the **PolicyCounterData** data structure to the callback address provide by the application.

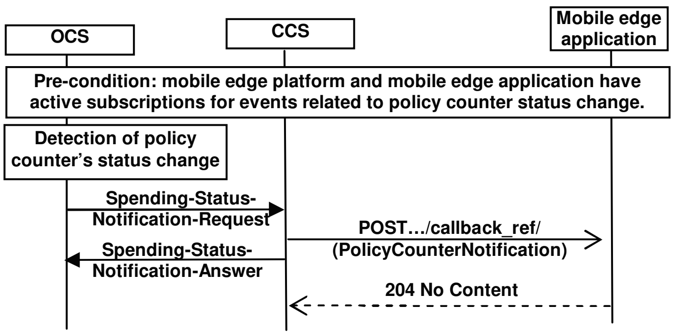
OCS
CCS
Mobile edge application
Pre-condition: mobile edge platform and mobile edge application have active subscriptions for events related to policy counter status change.
Detection of policy counter's status change
Spending-Status-Notification-Request
Spending-Status-Notification-Answer
POST.../callback_ref/ (PolicyCounterNotification)
204 No Content

Fig.7 Flow of notification about policy counter's status change

IV. DATA MODEL

This section describes the data model of resources representing the access to subscriber spending limits functionality.

The **PolicyCounterInfo** data type represents information about the status of policy counters that are associated with a specific mobile edge application instance. The attributes of **PCStatus** are as follows:

- **timeStamp** identifies when the policy counter status change occurred;
- **appInsId** uniquely identifies the mobile edge application instance;
- **requestId** uniquely identifies the request for the policy counter status information. It is allocated by the application;
- **userID** uniquely identifies the subscription of interest;
- **policyCounterList** is a structure of one or more **policyCounter** and indicates the list of policy counter identifiers to be subscribed to;
- **policyCounter** represents information about the policy counter status. It is a structure of **policyCounterID, policyCounterStatus,** and **pendingPolicyCounterInfo**;
- **policyCounterID** uniquely identifies a policy counter of interest;
- **policyCounterStatus** identifies the policy counter status applicable for the subscriber. The actual values are not specified, but an example values are provided in the next section;
- **pendingPolicyCounterInfo** contains the pending counter status and the active time. It is a structure of **policyCounterStatus,** and **pendingPolicyCounterChange-Time;**
- **pendingPolicyCounterChangeTime** indicates the expected time at which the pending policy counter status becomes the current status of the policy counter.

The **SubscriptionData** type represents a subscription to policy counter status change notifications from CCS. The attributes of **PCSsubscription** type are as follows:

- **callbackReference** is a URI (Uniform Resource Identifier) provided by the application, showing the place she wants to receive notifications;
- **filterCriteria** represents a list of filtering criteria for the subscription for policy counter status related events, which are also included in the response. It is a structure of **appInsId, userID,** and **policyCounterList** as defined earlier;
- **expiryDeadline** indicates when the subscription for policy counter status changes expires.

The **PolicyCounterNotification** type represents a notification about policy counter status change. It is a list of **policyCounter-Status.**

The proposed data model enables interoperability as it provides uniform access to the same data structure by different applications.

V. INTERFACE DEFINITION

The structure of resources, supported by the CCS, is shown in Fig.8. Each resource has a unique URI. All CCS resources have the following root:

{apiRoot}/ccs/{apiVersion}/

Following the RESTful architectural style, all resources are manipulated using four operations implemented by HTTP requests: POST, GET, PUT and DELETE. Table I represents the resources and the supported methods.

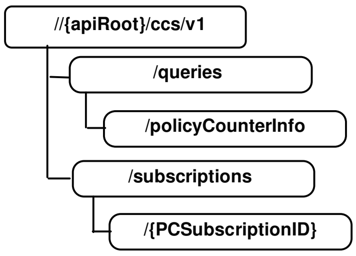

Fig.8 Structure of resources supported by CCS

TABLE I. CCS RESOURCES AND SUPPORTED HTTP METHODS

Resource name	Resource URI	HTTP method	Description
All queries about policy counter status	/queries	GET	Retrieves the list of all queries about policy counter state.
Policy counter status information	/queries/ policyCounterInfo	GET	Retrieves information about policy counter status.
All subscriptions for subscriber	/subscriptions	GET POST	Retrieves a list with all subscriptions related to policy counter status change. Creates a new subscription for policy counter status change
Existing subscription	/subscriptions/ subscriptionID	GET POST DELETE	Retrieves information about existing subscription for policy counter status change. Modifies existing subscription. Deletes existing subscription.
Notification callback	Callback reference provided by the application	POST	Sends a notification

VI. STATE MODELS

Implementation of the mobile edge CCS and a mobile edge application that make used of CCS API requires development of models, representing the resource state. The models representing the states related to subscriber spending limits supported by the CCS and by the application need to be synchronized.

Fig.9 shows a simplified model of the application view on the policy counter status.

A simple 4 level model related to policy counter state might be defined as: **valid, pending_invalid, invalid**, and **pending_valid**. In **valid** state, the subscriber has not reached the agreed spending limit. In **pending_invalid** state, the subscriber approaches the agreed spending limit threshold and the expected time, after which this threshold is expected to be reached, is **Δt**. For example, the subscriber is allowed to use higher data speeds in less busy periods and such a period is about to expire. In **invalid** state, the subscriber has reached the agreed threshould for the spending limit. In **pending_valid** state, the subscriber has reached the threshold related to the spending limit, but the expected time for the policy counter status to become **valid** is **Δt**. In **pending_invalid** state, the subscriber may recharge his account and the policy counter status becomes **valid**. For example, the busy period during which the subscriber is not allowed to use higher data speeds is about to expire. The transitions from **valid** to **invalid** directly or vice versa are possible due to administrative actions. In **unknown** state, the application does not have information about policy counter status. In **valid** and **invalid** states, the application may subscribe for notifications about policy counter status change, as well as to terminate the subscription.

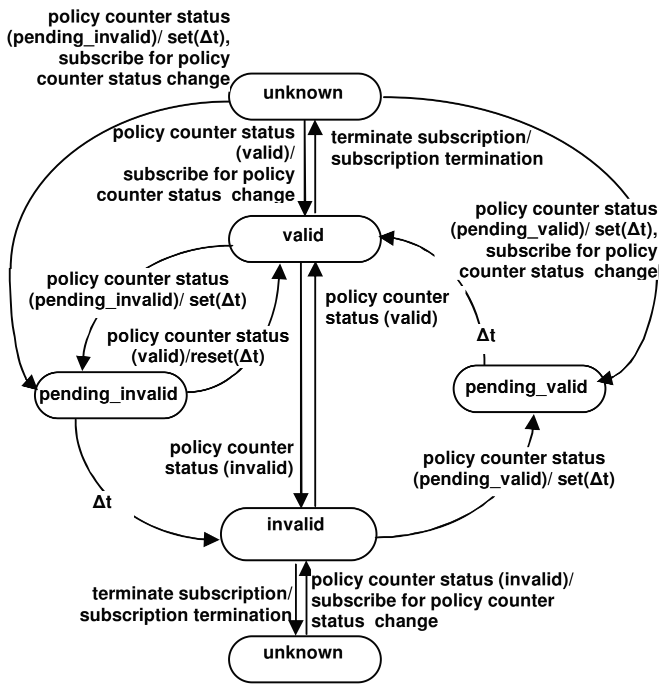
policy counter status (pending_invalid)/ set(Δt), subscribe for policy counter status change
unknown
policy counter status (valid)/ subscribe for policy counter status change
terminate subscription/ subscription termination
valid
policy counter status (pending_valid)/ set(Δt), subscribe for policy counter status change
policy counter status (pending_invalid)/ set(Δt)
policy counter status (valid)/reset(Δt)
policy counter status (valid)
Δt
pending_invalid
pending_valid
policy counter status (invalid)
policy counter status (pending_valid)/ set(Δt)
Δt
invalid
terminate subscription/ subscription termination
policy counter status (invalid)/ subscribe for policy counter status change
unknown

Fig.9. Model of policy counter state as seen by the mobile edge application

Fig.10 shows the model representing the tracking of spending limits for given subscriber, supported by the CCS.

In **Idle** state, the tracking of subscriber spending limits is not activated. In Idle state, the CCS may be asked about policy counter status. The mobile edge platform initiates a session with OCS (not shown in the figure), and retrieves the status of the policy counter requested. When the application subscribes for notifications related with changes in the policy counter state, it waits for changes in **WaitForPolicyCounterChange** state.

Both models are simplified; they represent only successful execution of the relevant procedures in the network and do not take into account abnormal conditions associated with unsuccessful procedures.

In order to prove in a mathematical manner that both models expose equivalent behavior we formalize the models' description. The notion of Labeled Transition System (LTS) is used to describe each model.

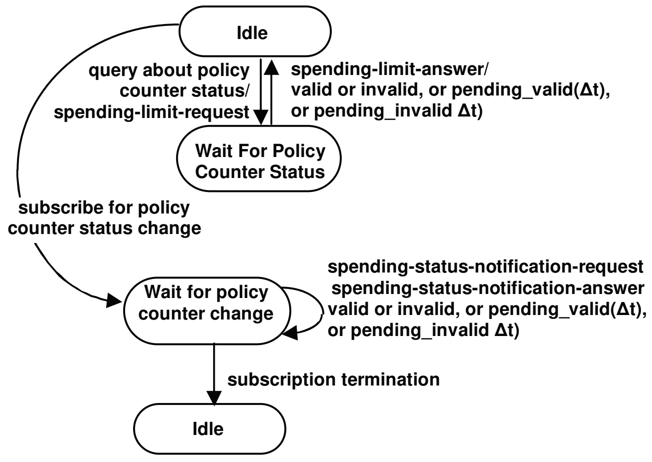
Idle
query about policy counter status/ spending-limit-request
spending-limit-answer/ valid or invalid, or pending_valid(Δt), or pending_invalid Δt)
Wait For Policy Counter Status
subscribe for policy counter status change
spending-status-notification-request spending-status-notification-answer valid or invalid, or pending_valid(Δt), or pending_invalid Δt)
Wait for policy counter change
subscription termination
Idle

Fig.10. Model, representing the spending limits tracking of given subscriber, supported by the CCS

A Labeled Transition System is represented as quadruple of a set of states, a set of actions, a set of transitions and a set of initial states.

By $T_{App} = (S_{App}, Act_{App}, \rightarrow_{App}, s_0^{App})$ it is denoted an LTS, representing the model of policy counter state as seen by the application, where:

- S_{App} = {Unknown [s_1^A], Valid [s_2^A], Pending_Invalid [s_3^A], Invalid [s_4^A], Pending_Valid [s_5^A]};

- Act_{App} = {policyCounterStatus(valid)[t_1^A], policyCounterStatus(pending_invalid) [t_2^A], policyCounterStatus(invalid) [t_3^A], policyCounterStatus(pending_invalid) [t_4^A], terminateSubscription [t_5^A], Δt [t_6^A]};

- $\rightarrow_{App}$ $= \{(s_1^A\ t_1^A\ s_2^A), (s_1^A\ t_2^A\ s_3^A), (s_1^A\ t_3^A\ s_4^A), (s_1^A\ t_4^A\ s_5^A), (s_2^A\ t_2^A\ s_3^A), (s_3^A\ t_1^A\ s_2^A), (s_3^A\ t_6^A\ s_4^A), (s_2^A\ t_3^A\ s_4^A),$

$(s_4^A\ t_1^A\ s_2^A), (s_4^A\ t_4^A\ s_5^A), (s_5^A\ t_6^A\ s_2^A), (s_2^A\ t_5^A\ s_1^A),$
$(s_4^A\ t_5^A\ s_1^A)\}$;

$s_0^{\mathrm{App}} = \{ s_1^A \}$.

Short notations for states and actions are given in brackets.

By $T_P = (S_P, Act_P \rightarrow_P, s_0^P)$ it is denoted an LTS, representing the model for spending limits tracking of given subscriber, supported by the CCS, where:

- S_P = {Idle [s_1^P], WaitForPolicyCounterStatus [s_2^P], WaitForPolicyCounterChange [s_3^P]};

- Act_P = { queryPolicyCounterStatus [t_1^P], spendingLimitAnswer [t_2^P], subscribeForPolicyCounterChange [t_3^P], spendingStatusNorificationRequest [t_4^P], subscriptionTermination [t_5^P]};

- $\rightarrow_P$ $= \{(s_1^P\ t_1^P\ s_2^P), (s_2^P\ t_2^P\ s_1^P), (s_1^P\ t_3^P\ s_3^P), (s_3^P\ t_4^P\ s_3^P), (s_3^P\ t_5^P\ s_1^P)\}$;

$s_0^P = \{ s_1^P \}$.

The synchronized behavior of both models is formally proved by using the concept of weak bisimilarity.

Intuitively, in terms of observed behavior, two LTSs are equivalent, i.e. they are bisimilar, if one LTS displays a final result and the other LTS displays the same result [19]. In practice, strong bisimilarity puts strong conditions for equivalence which are not always necessary. In weak bisimilarity, internal transitions can be ignored.

Proposition: T_{App}, and T_P are weakly bisimilar.

Proof: As to definition of weak bisimulation, it is necessary to identify a relation between the states of both LTSs, such as for any transition from a state in one LTS there are respective transitions from states in the other LTS.

By U_{AppP} it is denoted a relation between the states of T_{App}, and T_P, where U_{AppPe}= $\{(s_1^A, s_1^P), (s_2^A, s_3^P), (s_4^A, s_3^P), (s_2^A, s_1^P), (s_3^A, s_1^P), (s_4^A, s_1^P), (s_5^A, s_1^P)\}$. Then the following transitions for the states in U_{AppPe} are identified:

1. The mobile edge application makes a query about policy counter status and subscribes for changes in the policy counter state. The policy counter status is **valid**. For $(s_1^A\ t_1^A\ s_2^A) \exists (s_1^P\ t_1^P\ s_2^P), (s_2^P\ t_2^P\ s_1^P), (s_1^P\ t_3^P\ s_3^P)$.
2. The mobile edge application makes a query about policy counter status and subscribes for changes in the policy counter state. The policy counter status is **pending_invalid** and becomes **valid**. For $(s_1^A\ t_2^A\ s_3^A)$, $(s_3^A\ t_1^A\ s_2^A) \exists (s_1^P\ t_1^P\ s_2^P), (s_2^P\ t_2^P\ s_1^P), (s_1^P\ t_3^P\ s_3^P), (s_3^P\ t_4^P\ s_3^P)$.
3. The mobile edge application makes a query about policy counter status and subscribes for changes in the policy counter state. The policy counter status is **pending_invalid** and becomes **invalid**. For $(s_1^A\ t_2^A\ s_3^A)$, $(s_3^A\ t_6^A\ s_4^A) \exists (s_1^P\ t_1^P\ s_2^P), (s_2^P\ t_2^P\ s_1^P), (s_1^P\ t_3^P\ s_3^P)$.
4. The mobile edge application makes a query about policy counter status and subscribes for changes in the policy counter state. The policy counter status is **invalid**. For $(s_1^A\ t_3^A\ s_4^A) \exists (s_1^P\ t_1^P\ s_2^P), (s_2^P\ t_2^P\ s_1^P), (s_1^P\ t_3^P\ s_3^P)$.
5. The mobile edge application makes a query about policy counter status and subscribes for changes in the policy counter state. The policy counter status is **pending_valid** and becomes **valid**. For $(s_1^A\ t_4^A\ s_5^A)$, $(s_5^A\ t_6^A\ s_2^A) \exists$ $s_1^P\ t_1^P\ s_2^P), (s_2^P\ t_2^P\ s_1^P), (s_1^P\ t_3^P\ s_3^P)$.
6. The policy counter status is valid and changes to **pending_invalid** and then to **invalid**: For $(s_2^A\ t_2^A\ s_3^A)$, $(s_3^A\ t_6^A\ s_4^A) \exists (s_3^P\ t_4^P\ s_3^P)$.
7. The policy counter status is **valid** and changes to **pending_invalid** and then back to **valid**: For $(s_2^A\ t_2^A\ s_3^A)$, $(s_3^A\ t_1^A\ s_2^A) \exists (s_3^P\ t_4^P\ s_3^P)$.
8. The policy counter status is **valid** and changes to **invalid**: For $(s_2^A\ t_3^A\ s_4^A) \exists (s_3^P\ t_4^P\ s_3^P)$.
9. The policy counter status is **invalid** and changes to **valid**: For $(s_4^A\ t_1^A\ s_2^A) \exists (s_3^P\ t_4^P\ s_3^P)$.
10. The policy counter status is **invalid** and changes to **pending_valid**, and then to **valid**: For $(s_4^A\ t_4^A\ s_5^A)$, $(s_5^A\ t_6^A\ s_2^A) \exists (s_3^P\ t_4^P\ s_3^P)$.
11. The application terminates the subscription for policy counter status changes while the policy counter status is **valid**. For $(s_2^A\ t_5^A\ s_1^A) \exists (s_3^P\ t_5^P\ s_1^P)$.
12. The application terminates the subscription for policy counter status changes while the policy counter status is **invalid**. For $(s_4^A\ t_5^A\ s_1^A) \exists (s_3^P\ t_5^P\ s_1^P)$.

Therefore T_{App}, and T_P are weakly bisimilar, i.e. they expose equivalent behavior. ■

VII. Conclusion

Advanced charging capabilities in 5G system enable monitoring the usage limit (e.g. monetary, volume, duration) that a subscriber is allowed to consume. When the subscriber spending limit has reached a pre-set limit, the system is able to trigger a QoS downgrade and/or restrict access to one, several or all IP services based on operator pre-defined thresholds. When the subscriber spending limit has been increased, the system is able to modify resources (e.g. QoS, bandwidth, access) to services accordingly.

In this paper we propose a new mobile edge service that provides open access to monitoring the subscriber spending limits. Using the service API a mobile edge application may apply policy based decision for user traffic. The service is described by typical use cases, which illustrate its functionality, by data model representing the resource structure and the respective API definition. Some implementation aspects are discussed concerning modeling the behavior of the mobile edge platform and the generic logic of mobile applications using the service API.

Moving the policy-based control based on subscriber spending at the network edge provides more flexible charging capabilities close to the end user.

ACKNOWLEDGMENT

The research is conducted under the grant of project DH07/10-2016, funded by Bulgarian National Science Fund, Ministry of Education and Science.

REFERENCES

[1] E. Kapassa, M. Touloupou, A. Mavrogiorgou and D. Kyriazis, "5G & SLAs: Automated proposition and management of agreements towards QoS enforcement," *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, Paris, 2018, pp. 1-5.

[2] F. Z. Yousaf *et al.*, "Network slicing with flexible mobility and QoS/QoE support for 5G Networks," *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, Paris, 2017, pp. 1195-1201.

[3] E. Pencheva, I. Atanasov, "Usage Monitoring Control in Radio Access Network," *23rd Conference of Open Innovations Association FRUCT*, Bologna, Italy, 2018, pp.306-314.

[4] S. Guergov**.** Acupressure in magneto therapy environment, *Series on Biomechanics*, Bulgarian Academy of Science, vol.32, no.1, 2018, p. 16-19.

[5] S. Abdelwahab, B. Hamdaoui, M. Guizani and T. Znati, "Network function virtualization in 5G," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 84-91, April 2016.

[6] F. Z. Yousaf, M. Bredel, S. Schaller and F. Schneider, "NFV and SDN - Key Technology Enablers for 5G Networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468 - 2478, Nov. 2017

[7] D. Sinh, L. V. Le, L.P. Tung, B.S. P. Lin, "The Challenges of Applying SDN/NFV for 5G & IoT," *IEEE Vehicular Technology Society Asia Pacific Wireless Communications Symposium, APWCS'2017*, Incheon, South Korea, pp.1-6.

[8] Y. Lin, "Keynote topic: Network cloudification: SDN-NFV and 5G-MEC with edge and fog computing," *27th International Telecommunication Networks and Applications Conference (ITNAC)*, Melbourne, VIC, 2017, pp. 1-8.

[9] V. Sciancalepore, F. Giust, K. Samdanis, Z. Yousaf, "A double-tier MEC-NFV architecture: Design and optimization," *2016 IEEE Conference on Standards for Communications and Networking CSCN*, Berlin, 2016, pp. 1-6.

[10] Y. Nam, S. Song, J. Chung, "Clustered NFV Service Chaining Optimization in Mobile Edge Clouds," *IEEE Communications Letters*, vol. 21, no. 2, pp. 350-353, Feb. 2017.

[11] S. Song, J. Chung, "Sliced NFV service chaining in mobile edge clouds," *19th Asia-Pacific Network Operations and Management Symposium, APNOMS*, Seoul, 2017, pp. 292-294.

[12] ETSI GS MEC 012, *Mobile Edge Computing (MEC); Radio Network Information API*, v1.1.1, 2017.

[13] ETSI GS MEC 014. *Mobile Edge Computing (MEC); UE Identity API*, v1.1.1, 2017.

[14] ETSI GS MEC 002. *Mobile Edge Computing (MEC); Technical Requirements*, v1.1.1, 2016.

[15] 3GPP TS 23.203 Technical Specification Group Services and System Aspects; Policy and Charging Control architecture, 2018, Release 15, v15.3.0.

[16] 3GPP TS 29.219 Technical Specification Group Services and System Aspects; Service aspects; Charging and billing, Release 16, v16.1.0, 2018.

[17] 3GPP TS 29.219 Technical Specification Group Services and System Aspects; Policy and Charging Control: Spending Limit Reporting over Sy reference point, 2018, Release 15, v15.1.0.

[18] ETSI GS MEC 015. *Mobile Edge Computing (MEC); Bandwidth Management API*, v1.1.1, 2017

[19] X. J. Chen, R. De Nicola, "Algebraic characteristics of trace and decorated trace equivalences over tree-like structures," *Theoretical Computer Science*, 254, Elsevier, 2001, pp.337-361.