Integrated Network and End-host Policy Management for Network Slicing

Alexander Rabitsch*, Themistoklis Anagnostopoulos†, Karl-Johan Grinnemo*, Joseph McNamara‡, Anne-Marie Bosneag‡, Michail Alexandros Kourtis†, George Xilouris†, Özgü Alay§, and Anna Brunstrom*
*Karlstad University, Karlstad, Sweden
alexander.rabitsch@kau.se karl-johan.grinnemo@kau.se anna.brunstrom@kau.se
†NCSR Demokritos, Athens, Greece
thmanagnostopoulos@iit.demokritos.gr akourtis@iit.demokritos.gr xilouris@iit.demokritos.gr
‡Ericsson, Athlone, Ireland
joseph.mcnamara@ericsson.com anne.marie.cristina.bosneag@ericsson.com
§University of Oslo, Oslo, Norway
ozgua@ifi.uio.no

***Abstract*—5G mobile networks introduce the concept of network slicing, the functionality of creating virtual networks on top of shared physical infrastructure. Such slices can be tailored to various vertical services. A single User Equipment (UE) may be served by multiple network slice instances simultaneously, which opens up the possibility of dynamically steering traffic in response to the specific needs of individual applications – and as a reaction to events inside the network, e.g., network failures.**

This paper presents the PoLicy-based Architecture for Network Slicing (PLANS). In this policy framework, the network slice management entity in the 5G core and the UE can cooperatively optimize the usage of the available network slices via policy systems installed both inside the network and on the UE. The PLANS architecture has been implemented and evaluated in a 5G testbed. For two different case studies, we show how such a system can be leveraged to provide optimized services and increased robustness against network failures. First, we consider a drone autopilot scenario, and demonstrate how PLANS can reduce network-slice recovery time by more than 90%. Second, we illustrate for a 360°video streaming scenario how PLANS can help prevent video quality degradation when a network slice becomes unavailable.

***Index Terms*—5G, network slicing, transport services, user equipment, policy system**

I. Introduction

In a traditional mobile network, all communications must share the same resources. However, the requirements of some applications may conflict with each other. It is, therefore, very challenging to satisfy all requirements concurrently. Consequently, 3GPP has introduced the network slicing concept in 5G for managing such diverse requirements. The network slicing paradigm leverages virtualization techniques to create multiple, isolated, virtual end-to-end Network Slice Instances (NSI) on top of a shared physical network, whose properties can be tailored towards specific requirements.

At the heart of the network slicing architecture is the *slice manager*. This entity manages and monitors the network slices throughout their lifetime. The slice manager has a holistic view of the network, enabling it to effectively manage the network's resources while ensuring that the Quality of Service (QoS) requirements on the different NSIs are fulfilled. In order to effectively adapt to the dynamics of live networks, the slice manager must be able to monitor the health and status of its slices. In scenarios where a slice fails to live up to the agreed-upon service level agreements, including worst-case scenarios, such as when one or more network functions fail and connectivity is lost, the slice manager must take the appropriate actions to resolve the issue as soon as possible. Close coupling between the slice manager and the User Equipment (UE) services can help maintain connectivity, even in adverse scenarios.

This paper presents the *PoLicy-based Architecture for Network Slicing* (PLANS), an extensive policy framework for network slice management. We build upon our previous work [1], where we extended the network slice management concept to include the dynamic configuration of the network stack on the UE devices. In this work, we consider a more comprehensive, policy-based framework where policy engines installed inside the network and on the UE devices cooperate to flexibly manage the network's resources. In two separate case studies, we show how such a framework can benefit different applications: a low-latency drone control flow and 360°video streaming.

The rest of this paper is structured as follows: Section II provides a background on the network slicing concept and policy engines. Section III introduces the overall system design of the proposed framework. Section IV presents an example of how the proposed framework can be implemented in practice. Section V presents the two above mentioned case studies. Section VI discusses some considerations for implementing PLANS in 3GPP networks, and Section VII presents related work. The paper concludes with Section VIII.

II. Background

This section presents relevant background information on some of the key technologies behind the proposed system.

A. Network Slicing

The network slicing concept was conceived to satisfy an increasing emergence of various services with unique requirements, such as Industry 4.0, connected vehicles, e-Health, among many others.

An NSI is defined in [2] as a set of network functions and the resources for these network functions, which are arranged and configured to form a complete logical network that meets specific characteristics in terms of available bandwidth, latency, or QoS, among others described in 5QI (5G QoS Indicator). An NSI contains a list of Network Slice Subnet Instances (NSSI), which in turn are comprised of the Network Functions (NFs) and information about the interconnections between them, including the topology and individual link requirements (e.g., QoS attributes). The NSI is created from a NEtwork Slice Template (NEST).

The lifecycle of an NSI comprises four distinct phases: a preparation phase, an instantiation, configuration and activation phase, a run-time phase, and a decommissioning phase [3]. The slice manager, i.e., the Network Slice Management Function (NSMF), is responsible for managing the lifecycle of an NSI. The slice manager is ready to process incoming requests for communication services, provided that the preparatory tasks, such as network slice design and network slice pre-provisioning, have been done. When the slice manager receives a service request, it selects a NEST that can provide the agreed service requirements and commissions an NSI consisting of shared and/or slice-specific radio access network and core network functions. During the run-time phase, the slice manager creates performance management jobs for the NFs in each NSSI to generate performance data of a network slice instance and monitor thresholds for selected performance parameters.

B. Policy Engines

The PLANS framework integrates policy systems in the network and the end host. Therefore, as an introduction to policy systems, we present the APEX policy system, which is used inside the 5G core in the proposed framework. Next, we introduce the Transport Services architecture, which we use for policy management on the UE.

1) The APEX Policy System: The Adaptive Policy Execution Engine (APEX)[1] is a versatile and powerful policy engine that enables automatic optimization in 5G networks. APEX is an open-source tool that is part of the Open Network Automation Platform (ONAP) that offers an easy way to integrate with various systems requiring an automated decision-making element without the need to use any other ONAP components.

The APEX policy engine supports automatic decision-making by accepting input events and requests from other components, routing the input to the appropriate policies, computing the policy results, and generating response events/actions to be processed by other components. One characteristic

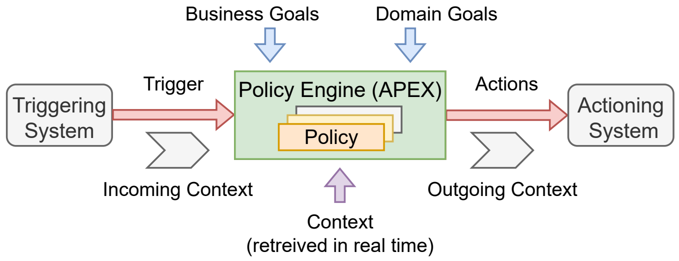

Fig. 1: The general APEX architecture.

of APEX is that it can handle adaptive policies, i.e., policies that can modify their behavior based on system and network conditions. Figure 1 presents the general APEX architecture.

The following are the main components of APEX: (i) the *Triggering System*, which receives events that can trigger a policy. The connection supports standard technologies, such as messaging systems, e.g., Kafka and Websockets, file input, and standard input, (ii) the *Actioning System*, which sends the result of a policy. Like the triggering system, the interface supports messaging systems and file output / standard output, and (iii) *Policies*, which are defined in a Universal Execution Policy Specification (UEPS). They are directly executable in an APEX engine. Higher-level policy specifications or existing policy languages can be easily translated into UEPS.

APEX is also able to provide *context* information for all events. Context information can be read from any outside source, and it is automatically shared between engine instances in an APEX system. Information injected into the policy context may impact the policies. This information may originate from business or domain goals, information derived from executions of the current policies, and context information retrieved from other components.

2) Transport Services: Transport Services (TAPS) [4] is an architecture that enables the decoupling of networked applications from the underlying network stack. TAPS defines a flexible, protocol-agnostic API, which enables an application to request an abstract *transport service* based on the application's high- or low-level requirements. The TAPS architecture is comprised of two parts: the *TAPS API* and the *TAPS Implementation*, as illustrated in Figure 2.

Before the connection establishment, the application describes its properties, i.e., its intent, its requirements, and its preferences, which may influence the selection of network path between the local and remote end-points, the selection and configuration of a transport protocol, as well as the security features for the connection. The properties passed from the application to the TAPS API are fed to a policy engine in the TAPS Implementation. In combination with policies and cached information, e.g., previous connections (which may be provided by an external system), the policy component uses the provided properties to generate a collection of candidate connections that best implement the service described by the application. The final transport service used for the communication is selected through a racing process [5]; a generalized version of Happy Eyeballs [6]. The first candidate to establish a connection to the remote host is then chosen for

[1]https://docs.onap.org/projects/onap-policy-parent/en/latest/apex/apex.html, Accessed on: February 2022.

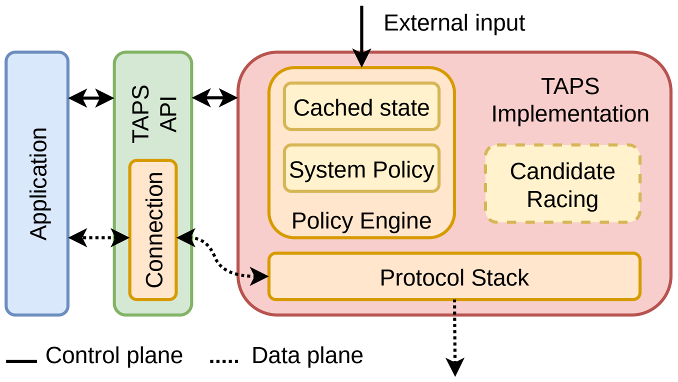

Fig. 2: The Transport Services architecture.

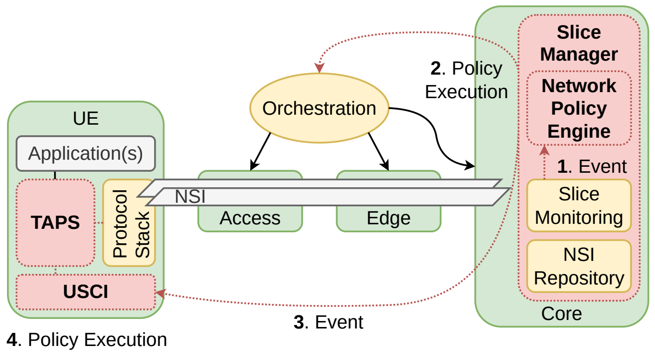

Fig. 3: An overview of the PLANS architecture. The building blocks and communication channels are highlighted in red.

the communication.

III. System Design

A 5G network may serve a single UE with up to eight NSI concurrently, which allows a UE to set up and dynamically steer traffic over several network slices in response to the specific needs of its hosted applications. PLANS is an architecture that enables user-equipment-assisted network slice selection to leverage this feature. The PLANS architecture is depicted in Figure 3. The slice manager is responsible for the lifecycles, i.e., the preparation, instantiation, runtime, and decommission of NSIs, which comprise the data plane in the access, edge, and core networks. The slice manager is also responsible for mapping NEST requests to one of the network slice types supported by the network slicing architecture and monitoring all active network slices. The slice manager enlists its internal network-side policy engine to optimize the usage of the available network slices, and to quickly react to detected network slice anomalies.

Upon detecting an anomaly, the slice manager informs the network policy engine of the event (1). The network-side policy engine may recommend that the slice manager perform a series of actions (2). Such actions may involve the orchestration layer, such as attempting to restart a failing Network Service (NS). Another action may be to signal the event to the UE devices connected to the anomalous slice (3). Such information is exchanged via a control plane channel between the slice manager and the UE Slice Configuration Interface (USCI) on the UE.

The USCI, in turn, supplies this information to the local policy unit of the TAPS implementation. The TAPS component assists and offloads UE-hosted applications to establish transport services that meet their QoS requirements. When a UE-hosted application requests a connection via the TAPS API, the TAPS policy engine directs communication to an appropriate slice based on the application properties, information about NSI properties and their current status, as well as pre-specified transport policies, e.g., the preferred use of particular NSIs over others. Learning about the failure of a network slice may for instance invoke a policy that directs communication to a backup slice (4).

Note that the whole communication chain does not necessarily need to be invoked – this depends on the context of the event. For non-critical scenarios, the initial action recommended by the network-side policy engine may be to restart a misbehaving NS, and thus stopping the communication chain after step 2. However, suppose the concerned slice has a history of failure; in that case, the policy may recommend a different action, such as temporarily steering the traffic over to a different slice and restarting the NS. In more critical scenarios, the policy may instead have the slice manager instruct the UE to immediately steer the traffic to another network slice at the first signs of failure. Once the NS has been restarted, PLANS may opt to move the traffic back to the original slice. Listings 1-2 show examples of network-side policies that may be triggered after an NS fails for the second time, and after the restart of said NS, respectively.

Listing 1: Failing slice example policy

```
{
  "event": {
    "type": "failingNS",
    "slice_id": <slice_id>
  }
  "policy": {
    "conditions": "failure_count": 1,
    "interal_actions": "restart_slice",
    "external_actions": {
      "notify_admin": false,
      "notify_ue": true
    }
  }
}
```

Listing 2: Slice restart example policy

```
{
  "event": {
    "type": "restartedNS",
    "slice_id": <slice_id>
  }
  "policy": {
    "external_actions": {
      "notify_admin": false,
      "notify_ue": true
    }
  }
}
```

IV. Implementation

In this section, we describe how we have implemented and integrated the main components within the PLANS architecture. We would like to stress that PLANS is generic, and the components do not rely on the specific implementations that we cover here.

A. The Katana Slice Manager

The Katana Slice Manager[2] is an open-source software component responsible for creating, managing, and monitoring 5G

[2]https://github.com/medianetlab/katana-slice_manager

network slice instances. It ties the platform coordination and the underlying infrastructure, management, and orchestration layers together. It communicates with the existing management and orchestration layer components through a South Bound Interface (SBI) to achieve this. Katana receives the NEtwork Slice Template (NEST) through a RESTful North Bound Interface (NBI). It provisions the slice, deploys the NS's, configures all the slice's physical and virtual elements, and activates the end-to-end operation.

The NEST is used for describing the parameters of a network slice. A NEST must complement the slice creation request a user sends to the Slice Manager to deploy a new slice. Katana's current implementation follows the NG.116 – Network Slice Template v2.0, defined by GSMA[3]. This document aims to assist network slice providers in mapping network slices' use cases into generic attributes. Katana parses the received NEST and, in conjunction with the supported capabilities of the underlying infrastructure, defines the virtual and physical functions that shall be deployed and configured as part of the slice.

The Katana Slice Manager is built as a mesh of integrated microservices working collectively to offer slice management services. Each service is implemented as an independent software module in an isolated container. The Slice Manager architecture exploits the microservices approach with all the communications between components on a message bus. The key advantages of this architectural approach are that it offers simplicity in building and maintaining applications, flexibility, and scalability, and the containerized approach makes the applications independent of the underlying system.

Further details about the Katana Slice Manager can be found in [7].

B. The APEX Policy System

The APEX engine has been integrated into the Katana slice manager software stack as a Docker container and enables automated decisions related to slice optimizations. The integration and workflow are shown in Figure 4. In our implementation, APEX receives triggering events from Prometheus, which runs as part of the Slice Monitoring module. When a specific part of a slice is failing or misbehaving, which results in a failed virtual NS, Prometheus generates an alert and sends it to the APEX policy engine via the internal Kafka message bus (message 1 in Figure 4). APEX processes the incoming alerts from Prometheus and recommends a series of appropriate actions to Katana. The recommended actions are based on the alert and the underlying system conditions and may depend on the use case/slice type. Therefore, once a trigger is received, a request is sent to get the policy context for this slice (messages 2-3). This context can include the history of issues that the slice has experienced and whether they have been resolved or not (message 4). This context information is used together with the trigger information to

[3]https://www.gsma.com/newsroom/all-documents/generic-network-slice-template-v2-0/

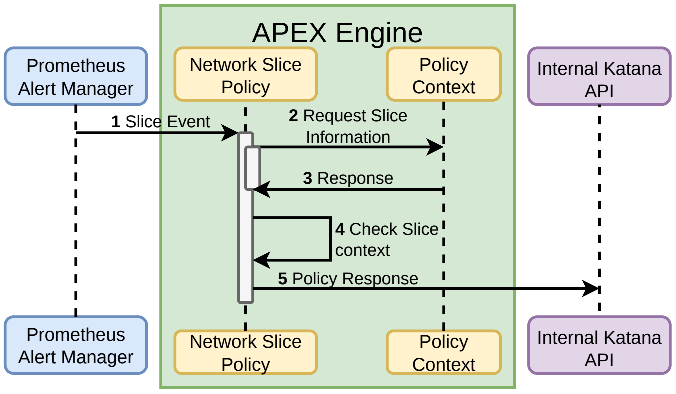

Fig. 4: A diagram for the integration of APEX and Katana.

generate the policy response, which is then sent over a REST API to the Katana slice manager (message 5), who will execute the recommended action.

C. The NEAT Module and the UE Slice Configuration Interface

On the UE, the policy system consists of two main components; the NEAT TAPS Module and the UE Slice Configuration Interface (USCI).

1) The NEAT TAPS Module: This work employs the *New, Evolutive API and Transport-Layer Architecture for the Internet* (NEAT) implementation of the TAPS architecture, which we deploy on the UE devices. A NEAT-enabled application may access the services of the NEAT module via the NEAT User API. NEAT's policy components are invoked at each new communication session. The *Policy Manager* implements the TAPS policy engine, the cached information is stored in a so called *Characteristics Information Base* (CIB), and the system policy in the *Policy Information Base* (PIB). Each policy consists of a set of rules that map requirements to a set of specific slices and protocol stack configurations. The policy manager uses the information in the PIB and the CIB to identify the most suitable network slice and protocol configuration that satisfies the application's requirements.

Applications on the UE may take advantage of a NEAT-enabled proxy that runs on the UE, which grants applications that do not natively support the NEAT API access to TAPS capabilities. The Linux *transparent proxy* feature is used to route traffic through the proxy, which terminates outgoing connections and then establishes a new connection to the original destination using the NEAT module.

2) The USCI: The USCI handles the communication between the Katana slice manager and the UE. When joining the network, the UE registers itself at the slice manager to subscribe to information on each connected slice. Once the registration process is done, the USCI begins to periodically poll for the latest information on the status of each slice. The polling frequency is in the order of tens of seconds, or even minutes, so as not to induce too much overhead on the network. In addition to the polling, the USCI also begins to listen for alert messages related to the subscribed slices sent from the slice manager. The information received from the slice manager, whether retrieved via polling or an alert message, is then prepared by the USCI for use in the NEAT module by adding the information to the CIB repository managed by the NEAT Policy Manager. Furthermore, the USCI also allows the slice manager to push policies to the UE, which may be combined with the information in the CIB to enforce a particular behavior from the UE.

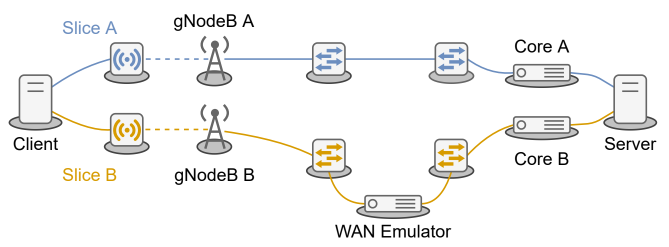

Fig. 5: The network topology employed in the testbed setup.

V. Case studies

To highlight the benefits of the PLANS architecture, we present two case studies, each showing a different use case. First, we introduce the testbed setup used in our experiments. Next, we present the two case studies: a low-latency drone control scenario, and a 360°-video streaming scenario. We have also evaluated the PLANS architecture in a bulk-transfer scenario, which we have left out for the sake of brevity.

A. *Testbed setup*

The PLANS architecture has been implemented in the 5GENESIS Athens platform [8]. Figure 5 depicts an overview of the network topology. Slices A and B utilize the Amarisoft Callbox SA system for enabling end-to-end 5G connectivity at the testbed, configured with different prioritization and QoS parameters to differentiate the communication capabilities of each slice. The client and server nodes are physical servers that reside at the edge and core locations of the Athens platform, respectively. The transport network connecting the two locations comprises multiple OpenFlow-based switches controlled by the Slice Manager to enable connectivity within the scope of the defined network slices. In addition to the physical SDN switches, a virtual WAN emulator based on Mininet is added as part of the Slice B path, allowing the creation of complex WAN topologies with various configurable QoS characteristics.

To illustrate the benefit of the proposed architecture, we use the communication channel between the UE and the slice manager to notify the NEAT module on the UE when a slice failure occurs. Using the information from Katana, the NEAT policy system can take immediate action and resolve the issue on the UE side instead of waiting until a failing slice returns to a normal and stable state. We repeat the experiments with the PLANS framework disabled as a baseline for comparison.

B. *Case study 1 - Latency sensitive drone control flow*

1) Experiment description: In this experiment, we replay a packet trace from a latency-sensitive flow between the client and the server captured from the PixHawk 4.0 drone autopilot. The capture comprises MAVLink packets sent over the 5G Radio Access Network (RAN) with navigation commands. We employ the QUIC protocol and its datagram extension to send the MAVLink packets and measure the latency between the server and the UE. We trigger a failure on the primary slice twice throughout the experiment. Since Slice A is optimized for low latency, it is desired to let the MAVLink flow use this slice as much as possible. The UE should be instructed to steer the flow to Slice A as soon as it becomes available, only resorting to Slice B if it is the only option. Therefore, we feed APEX with the following policy:

(i) If a slice has no previous history of failure, the slice manager attempts to correct the issue by restarting the misbehaving NS and notifying the NEAT policy system on the UE of the failure. In doing so, the NEAT module can take immediate action by switching over to the other slice; it directs the UDP packets that carry the QUIC flow over the new slice. When the slice manager has resolved the issue, NEAT is again notified.

(ii) If the slice has a history of failure, the slice manager is instructed to notify the system administrator as well as the NEAT module on the UE and take down the slice, and in so doing, allow the administrator to inspect the error.

2) Results: Figures 6 and 7 show the measured latency in two example runs; the first is the baseline with PLANS disabled and the second is with PLANS enabled. Although Slice B is not optimized for low latency, Figure 7 clearly illustrates the benefit of instructing the UE to temporarily switch over the latency-sensitive traffic to a secondary slice during an outage on the primary slice. The time required to correct a failing slice varies but can take more than 100 s as seen in Figure 6, i.e., an unacceptably long delay in a use case such as this. Switching over to a secondary slice as soon as the failure is detected during the outage of the primary slice maintains the connectivity, though, in this case, with a lower QoS. We want to emphasize that the lower QoS is due to the particular configuration of the secondary slice and not our proposed mechanism. Figure 8 shows the aggregate of the results in the form of an ECDF plot. Rather than omitting lost datagrams, they have been assigned a very long delay. As Figure 8 tells, around 20% of the datagrams arrive late (or not at all), even with PLANS enabled. This is because it still requires some time for the system to detect that Slice A is failing, and any datagram frames sent during this time are likely to be lost. However, compared to the baseline results, where less than 50% of the datagrams arrive in time, this is still a massive improvement.

C. *Case study 2 - 360°video streaming*

1) Experiment description: In this experiment, we emulate the playback of a 360°video on the UE, which experiences a failure on Slice A after 45 s. We use *dashc* as a DASH video client, which streams a video hosted on a DASH HTTP

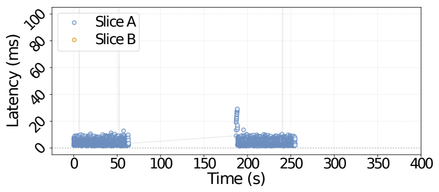

Fig. 6: Baseline drone results with the PLANS framework disabled.

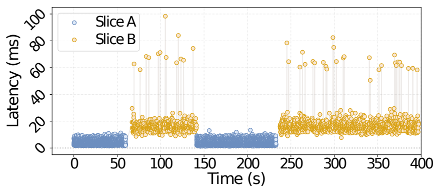

Fig. 7: Drone results with the PLANS framework enabled.

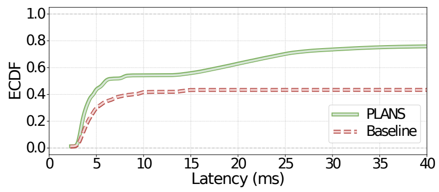

Fig. 8: Latency measured over the two slices, with and without PLANS.

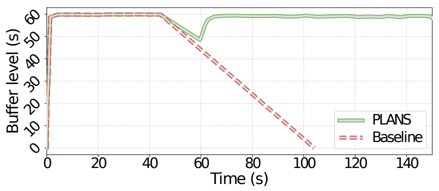

Fig. 9: The 360°video stream buffer levels, in seconds, over time.

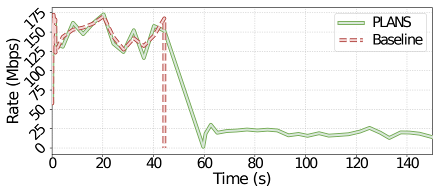

Fig. 10: The 360°video stream delivery rate over time.

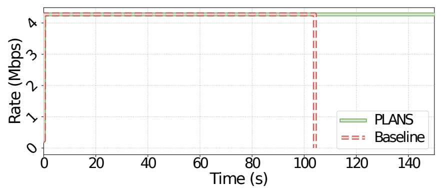

Fig. 11: Representation rate over time.

ngnix server. This setup enables us to measure various QoE metrics, such as the buffer level after each segment download, the representation rate (which maps to the video resolution), and the delivery rate of each segment. Both slices can provide the desired service in this use case, so there is no need to switch back to Slice A once it restarts after a failure. Therefore, we use the following APEX policy:

(i) If the slice experiences a failure, the slice manager will restart the misbehaving NS and instruct the NEAT policy system on the UE to switch to the other slice. In this case, the slice failure triggers a new TCP connection that is opened over the new slice.

2) Results: Figure 9 shows the buffer level over time in two example runs, with PLANS disabled and enabled, respectively. The buffer level is limited to 60 s. For the baseline without PLANS, the video stalls indefinitely after the failure on Slice A, whereas the video playback is unaffected when PLANS is enabled: The buffer is only drained during the 10 s until the switch to Slice B is made.

Both slices are capable of delivering the video at maximum quality. Nevertheless, the delivery rate of each segment, presented in Figure 10 is much more dynamic when PLANS is enabled, clearly showing the difference in the available capacity of the two slices. Still, the representation rate is not affected by whether or not the PLANS framework is enabled or disabled. As shown in Figure 11, the representation rate is consistently at the maximum, i.e., a bit above 4 Mbps, until it stalls in the baseline run. Due to the stall in the baseline experiments, which is not observed with PLANS, we conclude that the end-user experience is not affected by the slice failure when the PLANS framework is enabled. In other words, the transition between the slices when using PLANS is seamless, provided the buffer is large enough to cover the drainage during the switch to Slice B.

Since the video playback permanently stalls for the baseline runs when the TCP connection over slice A breaks, it is impossible to plot the aggregate results accurately. The appearance of such a plot would depend on factors such as the total length of the video and the choice of when the network slice failure on Slice A is initiated. Nevertheless, we consider the example results in Figures 9-11 representative of the collected dataset.

VI. Considerations for 3GPP Networks

For a commercial 5G network, a direct communication channel between the slice manager and the UE would seem unorthodox. 3GPP specifies no such interface between the UE and the slice manager. While there may be multiple possible solutions to allow the UE and the slice manager to communicate, one such solution is to use the Network Exposure Function (NEF) [9]. The NEF is a network function responsible for securely exposing network capabilities, events, and monitoring/analytics information, e.g., slice monitoring data, to one or more so-called Application Functions (AFs). An AF is an internal, trusted, or external, third-party network function that interacts with the 3GPP Core Network. Trusted AFs may interact with other NF directly, whereas third-party AFs must interact with other NFs indirectly via the NEF. The NEF acts as an intermediary between the network and the AFs, granting the AFs access to the capabilities provided by other 3GPP NFs. Typically, an AF is co-located with an Application Server, which may be reachable from the UE. Thus, a logical communication channel between the UE and the slice manager can be established by using the NEF. The UE would subscribe to events concerning the slices it connects to at the AS/AF and fetch up-to-date information on the slice from the slice manager via the NEF.

VII. Related work

To our knowledge, few works consider end node- or user equipment-assisted network slice selection in 5G networks. The work in [10] [11] is perhaps most closely related to our work. This work proposes a concept called *trailer slicing*, where packets generated by an application on a UE carry trailer bits containing metadata about their contents. These bits may be used by switches in the network to allocate appropriate network resources. Since this work precedes 5G, it primarily targets SDN/NFV networks in general and virtual network operators in 4G/LTE networks in particular. An example of a more recent work that explicitly targets slicing in 5G networks is the machine learning-based classification mechanism in [12] that the authors name *progressive slicing*. In [13], the authors propose a neural network-based, equipment-assisted network slice selection mechanism called SMASH. Also worth noting are the works on intent-based network-slice selection that have been proposed in recent years [14]–[17], which emanate from the large body of work conducted on policy- and intent-based networks [18]. These works suggest translating high-level service requirements into low-level network-slice configurations using an intent policy engine. The PLANS architecture distinguishes itself from the works described above by being the only solution that relegates the network slice selection at the end node to a local transport service or TAPS component. Early work on end-to-end network slicing and slice selection was conducted in [19]. This work was among the first to showcase the feasibility of using SDN, NFV, and software-defined radio to realize isolated end-to-end network slices over a cellular network. A more recent solution was proposed in [20], which employs deep reinforcement learning to efficiently control the selection of network slices in both the RAN and the core networks. The PLANS framework is distinguished from these solutions by the direct involvement of the UE in the network slice selection process. Notably, it selects network slices through the cooperation between the UE and the slice manager entity itself, thus leveraging the unique knowledge and capabilities of both the end-host and the network to select the most suitable slice for an application. In combination with the capabilities displayed in our previous work [1], this allows for better flexibility and utilization of the network's resources. Several works pertain to network slicing, although not directly considering the selection of network slices, e.g. [21], [22].

VIII. Conclusion

In this paper, we have presented PLANS, a framework in which policy systems in the 5G core and on a UE may cooperate to optimize the selection and steering of single application flows over network slices to meet their service demands. PLANS offers additional flexibility and robustness compared to existing 5G network slice management in that a UE and a slice manager can cooperatively ensure connectivity during the failure of a network slice. We have implemented PLANS in a 5G testbed and show in two case studies how PLANS makes it possible to keep a service running (albeit possibly with a lower QoS) during a network-slice failure, and thus show that PLANS or a similar integrated UE-5G core solution could be a solution to the problem of upholding application-specific service demands in current and future 5G networks.

References

[1] A. Rabitsch *et al.*, "Extending network slice management to the end-host," in *Proceedings of the 1st Workshop on 5G Measurements, Modeling, and Use Cases*, ser. 5G-MeMU '21. ACM, 2021, p. 20–26.

[2] 3GPP, "Technical Specification Group Services and System Aspects; System architecture for the 5G System (5GS) (Release 17)," 3GPP, Technical Specification (TS) 23.501, 09 2021, version 17.2.0.

[3] ——, "Technical Specification Group Services and System Aspects; Telecommunication management; Study on management and orchestration of network slicing for next generation network (Release 15)," 3GPP, Technical Report (TR) 28.801, 1 2018, version 15.1.0.

[4] T. Pauly *et al.*, "An Architecture for Transport Services," IETF, Internet-Draft draft-ietf-taps-arch-13, Jun. 2022, work in Progress.

[5] A. Brunstrom *et al.*, "Implementing Interfaces to Transport Services," IETF, Internet-Draft draft-ietf-taps-impl-13, Aug. 2022, work in Progress.

[6] D. Schinazi and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency," RFC 8305, Dec. 2017.

[7] T. Anagnostopoulos *et al.*, "Deliverable 3.4: Slice Management (Release B)," 5GENESIS, Tech. Rep., 2021. [Online]. Available: https://5genesis.eu/wp-content/uploads/2021/05/5GENESIS_D3.4_v1.0.pdf

[8] G. Xilouris *et al.*, "Deliverable 4.3: The Athens Platform (Release C)," 5GENESIS, Tech. Rep., 2021. [Online]. Available: https://5genesis.eu/wp-content/uploads/2021/09/5GENESIS-D4.3_v1.0.pdf

[9] 3GPP, "Network Exposure Function Northbound APIs; Stage 3, (Release 17)," 3GPP, Technical Specification (TS) 29.522, 06 2021, version 17.2.0.

[10] A. Nakao, "Application specific slicing for MVNO through software-defined data plane enhancing SDN," in *2016 Optical Fiber Communications Conference and Exhibition (OFC)*, 2016, pp. 1–2.

[11] A. Nakao and P. Du, "Application-specific slicing for MVNO and traffic characterization [Invited]," *Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. A256–A262, 2017.

[12] T. Iwai and A. Nakao, "Progressive Slicing for Application Identification in Application-Specific Network Slicing," in *GLOBECOM 2020*, 2020, pp. 1–6.

[13] A. Fontana *et al.*, "SMASH: a SMArt Slicing Heterogeneous 5G network selection algorithm," in *2020 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2020, pp. 1–6.

[14] K. Abbas *et al.*, "Network Slice Lifecycle Management for 5G Mobile Networks: An Intent-Based Networking Approach," *IEEE Access*, vol. 9, pp. 80 128–80 146, 2021.

[15] N. Gritli *et al.*, "Decomposition and Propagation of Intents for Network Slice Design," in *2021 IEEE 4th 5G World Forum (5GWF)*, 2021, pp. 165–170.

[16] T. A. Khan *et al.*, "Intent-Based Orchestration of Network Slices and Resource Assurance using Machine Learning," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–2.

[17] D. Wang *et al.*, "An Intent-based Smart Slicing Framework for Vertical Industry in B5G Networks," in *2021 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, 2021, pp. 389–394.

[18] E. Zeydan and Y. Turk, "Recent Advances in Intent-Based Networking: A Survey," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, 2020, pp. 1–5.

[19] C. Bektas *et al.*, "Towards 5G: An Empirical Evaluation of Software-Defined End-to-End Network Slicing," in *2018 IEEE Globecom Workshops (GC Wkshps)*, 2018, pp. 1–6.

[20] Q. Liu and T. Han, "When Network Slicing Meets Deep Reinforcement Learning," ser. CoNEXT '19 Companion. ACM, 2019, p. 29–30.

[21] C. Marquez *et al.*, "How Should I Slice My Network? A Multi-Service Empirical Evaluation of Resource Sharing Efficiency," ser. MobiCom '18. ACM, 2018, p. 191–206.

[22] D. Harutyunyan *et al.*, "Orchestrating End-to-end Slices in 5G Networks," in *2019 15th International Conference on Network and Service Management (CNSM)*, 2019, pp. 1–9.