5G Smart Devices Supporting Network Slicing White Paper

by NGMN Alliance

Version:	**1.1**
Date:	**15-December-2020**
Document Type:	Final Deliverable (approved)
Confidentiality Class:	P - Public

Project:	**Network Slicing Requirements for Operating Systems Of 5G Smartphones**
Editor / Submitter:	**Danni Song (China Mobile)**
Contributors:	**Jonathan Borrill (Anritsu), Aitong Han (CMCC), Chenguang Jin (CMCC), Danni Song (CMCC), Haiyu Ding (CMCC), Nan Li (CMCC), Wei Deng (CMCC), Xiaoxiong Song (CMCC), Changsoon Choi (DT), Per Synnergren (Ericsson), Stan Wong (HKT), Binjun Liu (Huawei), Fusheng Shen (Huawei), Xiaoliang Yu (Huawei), Gary Li (Intel), Chia-Yu Ku (MediaTek), Dustin Fan (MediaTek), Fangyi Huang (MediaTek), Antoine Mouquet (Orange), Hong Qiu (Qualcomm), Suli Zhao (Qualcomm), Yue Yin (Qualcomm), Arvin Siena (PLDT/Smart Ph), Timothy Senathirajah (PLDT/Smart Ph), Camillo Carlini (TIM), Davide Musella (TIM), Brett Christian (T-Mobile USA), Nick Baustert (T-Mobile USA), Scott Probasco (T-Mobile USA), Miao miao (Unisoc), Zhiwei Fu (Unisoc), Dingyuan Tu (ZTE)**
Approved by / Date:	**NGMN Board, 8th January 2021**

Address:

ngmn e. V.

Großer Hasenpfad 30 • 60598 Frankfurt • Germany

Phone +49 69/9 07 49 98-04 • Fax +49 69/9 07 49 98-41

Abstract: Short introduction and purpose of document

The 5G network system can provide on-demand network slices to satisfy different service requirements. However, how to create traffic connection for network slicing in 5G devices internally is still a technical gap.

The design of Network Slicing function in 5G devices has to rely on 5G devices operation system as well as the traffic descriptors of the service between the upper layer and the modem, which results in that the current 5G devices does not support the use of network slicing. Therefore, the fundamental purpose of this document is to provide the reference design of network slicing solution in the 5G devices.

Contents

1 INTRODUCTION

As a milestone in the development of communication technology, 5G network is expected to realize significant digital transformation to enable of a series of emerging Internet applications, and provide strong information service capabilities for individuals and enterprises. At present, the industry generally believes that 5G should not only have advanced wireless access technology, but also have flexible, open and customizable network service capabilities, so that the operators can provide differentiated network services such as large bandwidth, low delay, high reliability and massive connection to individual users or industry users according to the service requirements.

In 5G era, there will be a large number of devices connected to the network. These devices can belong to different industrial fields, and their requirements are different in network mobility, security, delay, reliability, and even billing methods. If we follow the traditional network construction ideas to meet the huge differences in different requirements only through a large homogeneous network, then for operators, it will cause a huge cost and inefficient investment; for users, we cannot get high-quality of services to meet specific needs. In this regard, 5G network slicing technology is considered to be the key to meet the diversified services requirements of 5G.

With the introduction of network slicing technology, operators will be able to provide network capabilities with different functional characteristics, which will provide "exclusive" network for users with different KPI requirements to ensure a high-quality of service and meet differentiated scenario requirements. And users can also use more cool application products, which will further stimulate the development of new industry applications. Finally, it achieves the goal of improving the efficiency of network resource utilization, optimizing the network construction investment of operators, and building a flexible and agile 5G network.

As the starting point of network slicing service, the introduction of network slicing features in the 5G devices also has a wide and significant impact on the applications, operating systems, communication chips and so on. But at present, the industry is still in the initial stage of research and exploration on how to realize the network slicing in the 5G devices, and no clear and unified understanding and consensus have been formed.

This white paper focuses on the application of network slicing in 5G devices, and analyzes the unique technical capability and service advantages of network slicing service. In this white paper, 5G devices include smartphones and user devices which include an operating system inside. Through the research and analysis of the key parameters and signaling messages of network slicing, combined with the actual design capability of the current system, this paper introduces the challenges faced by the characteristics of network slicing in the design and technical implementation of system. This paper introduces a variety of reference architectures and technical design schemes for network slicing in the devices, and proposes that 5G devices should support "the target scheme of network slicing in the devices" and "modem centralization scheme", which provides an industrial guidance for 5G devices to support network slicing capability.

2 SERVICE CAPABILITY OF NETWORK SLICING IN 5G DEVICE

2.1 Typical Use Cases and Scenarios of Network Slicing in the 5G device

In the 5G era, mobile networks need to serve devices of various types and requirements in terms of mobility, billing, security, policy control, latency, reliability and so on. Typical 5G services include cloud gameing, live video, high-definition low latency video and other services that are yet to be invented.

2.1.1 Cloud Gaming

2.1.1.1 Traffic Features of Cloud Gaming

Cloud gaming is based on the real-time audio and video streaming technology. The server sends real-time audio and video streams to the client, and the client sends a flow of control instructions to the server; then, the server applies the received control instructions to the game. The whole loop involves: game rendering in the server, audio and video coding, network transmission, audio and video decoding and rendering in the client. All of these processes are time-consuming to a certain extent, and the total time taken by this loop has the greatest impact on user experience.

The specific scenarios of cloud gaming put forward higher requirements to the network: Low latency, high bandwidth, no (small) jitter.

2.1.1.2 Network Slicing Requirement of Cloud Gaming

Low latency requirements for cloud gaming. If the existing 4G network and Wi-Fi are used, the response delay of cloud gaming fluctuates in the range of about 50ms ~ 140ms in normal network conditions. For some games with high requirements of operation response delay (for example: action, fighting, MOBA, shooting, racing, etc.), there is still a significant experience gap compared with local games, and operation response delay needs to be further reduced. Due to the particularity of its head-mounted display, VR devices can cause motion sickness when the latency exceeds 20ms to effectively control rejection.

High bandwidth requirements for cloud gaming. With current high-definition cloud games (1080P 30fps) in case of H264 encoding, the bit rate is about 8000Kbps, and the required network bandwidth is at least 10Mbps to run stably and smoothly. As application devices expand to large-screen TVs, VR devices, etc., higher resolutions (4K or even 8K) and frame rates (60fps or even 120fps) need to be provided, and the bit rate also increases. According to preliminary estimates, the 4K 60fps ultra-high-definition cloud games (using H264 encoding) will have a bitrate of about 60Mbps to 70Mbps, and at least 80Mbps of network bandwidth will be required to run stably and smoothly.

Low jitter requirements for cloud gaming. The biggest difference between cloud gaming and other audio and video streaming media is that it requires very low operation response delay (millisecond level), and there must be no or minimized buffered streaming media frame data. Because there is no buffered frame, every network jitter will cause the game to freeze, which puts forward higher requirements on the stability of the network.

Only when these network conditions are met, cloud gaming can bring users an experience comparable to the local games. 5G edge computing and network slicing are the best solutions to these problems. Edge computing has narrowed the distance between the end user and the server in the physical space, creating a basic condition for low latency. Network slicing is further refined in terms of functions, and a customized network environment can be realized to meet the network requirements of low latency, high bandwidth, and jitter-free according to the network requirements of cloud gaming scenario.

2.1.2 Live Video

2.1.2.1 Traffic Features of Live Video

The real-time return of media collected and edited images requires very high network uplink bandwidth. With the help of 5G network slicing services, it is possible to access dedicated network slicing protection and isolation according to the device, provide end-to-end dedicated lines and dedicated networks, meet the large uplink bandwidth requirements of media live broadcast scenarios, and provide high reliable and low-latency services based on isolation between slices to ensure data security. It can solve the current bottleneck

problems of high cost and insufficient flexibility caused by the return of important news events or events that can only be transmitted via satellite or optical fiber. For such services, the differentiated requirements of customers are mainly uplink bandwidth and delay requirements.

2.1.2.2 Network Slicing Requirement of Live Video

With the rapid development of internet live broadcast services, more and more users carry out live broadcast service and the demand of network uplink bandwidth is also increased accordingly. Especially for VR live broadcast and other new technologies, it strengthens the interaction of live broadcast and makes the audiences have a stronger sense of substitution and immersive experience. The audience can break through the limitations of the previous plane live broadcast and choose the viewing angle according to their preferences, so as to realize the immersive and high-definition virtual experience. These new technologies improve the requirements of network uplink and downlink bandwidth and network delay. 5G's eMBB slice or a dedicated slice will guarantee these new businesses to realize new technology elements and immersive user experience.

Table 2-1 SLA requirements of Live Video scenario

scenario	Uplink Guarantee Date Rate (Mbps)	Downlink Guarantee Date Rate (Mbps)	Network delay (ms)	Reliability	mobility
Live Video	100	50	100	99.5%	Low speed

2.1.3 HD video playback

2.1.3.1 Traffic Features of HD video playback

With the development of video technology, the industry video application has witnessed explosive growth. High definition video is integrated with industry scenes to serve all walks of life. 5G combined with video technology, not only makes video business become the basic business of operators, but also further promotes the deep integration of video and industry scene, showing the development trend of high-definition, mobile, intelligent and industrial.

5G network enables the ultra-high definition video device to break the restrictions of space, time and region. It can not only enable users to enjoy UHD video service anytime and anywhere, but also provide personal video live service for users anytime and anywhere. With 5G network slicing service, it can access proprietary slice protection and isolation according to the device, provide end-to-end dedicated line and private network, meet the requirements of large downlink bandwidth of live media scenes, provide high reliability and low delay services, and ensure the security of data based on the isolation between slices, so as to meet the development of HD video playback scenes.

2.1.3.2 Network Slicing Requirement of HD video playback

UHD video playback, such as 4K/8K video and UHD VR/AR, requires more than 100Mbps of real-time transmission bandwidth. The pixel of 4K UHD video is 4 times higher than 2K, and 8K is 4 times higher than

4K. 5G technology can effectively solve the problem of video jam and unclear picture caused by the ultra-high definition video due to network reasons, which can ensure the transmission quality of image and sound for online conference. Generally speaking, VR and AR technologies will correspond to two different devices. VR devices have higher requirements for delay, but also have certain requirements for bandwidth; AR devices have higher requirements for bandwidth, but also need to guarantee a certain delay. In terms of the requirements for VR and AR devices, UHD VR/AR not only has high requirements for downlink bandwidth, the network delay also needs to be minimized to 10ms-20ms while ensuring data transmission, in order to prevent users from dizziness caused by network delay. Through 5G network slicing technology to ensure transmission, it can fully meet the user experience in VR / AR applications, and bring users wireless immersive entertainment services anytime and anywhere.

Table 2-2 SLA requirements for HD video playback (combined with interaction)

scenario	Uplink Guarantee Date Rate (Mbps)	Downlink Guarantee Date Rate (Mbps)	Network delay (ms)	Reliability	mobility
VR/AR	50	150	20	99.5%	Low speed

2.2 Limitations of QoS Service Capabilities

Network slicing is sometimes compared with the Quality of Service (QoS) when talking about how communication network could provide differentiate service quality to users. As QoS has long history and is maturely in commercialization, network slicing is often been challenged on its necessity. Generally speaking, both network slicing and QoS can differentiate communication services quality offered, but the meanings of service quality are quite different.

QoS focuses on service quality. Refer to the 5QI definition in 3GPP, the service quality is measured by priority, packet delay budget, packet error rate, default maximum data burst volume, and default averaging window [3GPP TS23.501 Table 5.7.4-1]. In the 3GPP specification, there is service examples for each 5QI, but no accurate definition is provided on what the service is.

Take the 5QI value of 4 as an example (see table x below). Its packet delay budget is 300ms, the target packet error rate is 10^{-6}; the priority, maximum data burst volume, averaging window are configurable; one example services is non-conversational video. While other aspects that would impact user experience/service experience are not addressed which might be a key to have the traffic using the same 5QI be de facto different services.

Table 2-1 Example of Standardized 5QI

5QI Value	Resource Type	Default Priority Level	Packet Delay Budget	Packet Error Rate	Default Maximum Data Burst Volume	Default Averaging Window	Example Services
4	GBR	50	300 ms	10^{-6}	N/A	2000 ms	Non-Conversational Video

Compared to QoS, network slicing can reflect the differences not only the 5QI but also the meaningful services, thus differentiated network centric applications comprehensively. For example, for the same non-conversational video traffic stream, the 2K video streaming and 8K video streaming result in two differentiated service experiences. Although both streams reserve 5QI value of 4, but the two “services” are significantly different, and the network capabilities and resources required to support the two streams to achieve the desired qualities are significantly different. Along with the rich types of cellular devices, the different video resolutions and frame rate of a video streams will result in many “types” of “services”. With 5QI, it might help on differentiating the services by maximum data burst volume and averaging window, there would be network resource competition and confliction, so the network application experiences could not be deterministically guaranteed.

With the development of cloud technology, more and more applications have been powered by cloud. Applications on device have significant different requirement on their communication network. There are health monitoring applications with lower requirement on peak data rate, as well as interactive multimedia applications with higher interaction and high data throughput requirements. Furthermore, along with 5G device devices moves from consumer segment to vertical segment, the "services" accessed on 5G devices for cloud applications will be further enriched, and the differentiation of "service" will gradually be reflected in the commercial mobile network.

In order to meet the diverse communication requirements from cloud applications, and the characteristics of communication demand changing over time, mobile operators need to perform fine management and maintenance on network resources to build "customized" networks to meet the “services”. Network slicing will leverage virtualization techniques to virtually or physically isolate network slices, thus providing non-interfered communication networks for different "services".

By dividing network resources into multiple network slices, 5G network slices can provide services for industries with different requirements (such as delay, reliability, capacity, isolation, and other functions). Operator networks will not only serve the users with "Best Effort " transmission, but also serve users with "deterministic " transmission. The network can allocate logically or physically isolated network resources for these businesses with different communication requirements. The network resources include not only 5G core network control plane and user plane network functions, but also access network and transport network resources.

Moreover, network operators can implement the management and opening strategy separately for each network slice. The providers and users of cloud services/cloud applications can choose the network management strategy suitable for their own needs to deploy their "services". Through network slicing, operators can modularize their network function design to meet the needs of different communication services.

In summary, although both QoS based on 5QI and network slicing provide experience differentiation, network slicing focuses more on service differentiation. In the early days of mobile communication technology, the computing power of communication devices was not sufficient, so the available communication network "services" and their contents were not rich, and the difference was mainly based on quality. While with the enhancement of the computing power of communication devices and the development of cloud technology, the communication "services" that can be used by devices are becoming more and more abundant. Network slicing technology implements virtual networks on a single physical network and supports multiple network slices on the device side, which is an important technical solution for 5G devices to provide differentiated "services" for end users.

2.3 Advantages of Network Slicing Service Capabilities

As a means of providing network services, network slicing is often compared with traditional QoS.

Compared with QoS, network slicing can not only provide differentiated quality of service based on the different services, but also provide differentiated user experience for different users for the same service.

As the 5G era is an era of Internet of everything, access to a large number of different types of devices will generate diversified requirements, such as: services requiring large bandwidth and high speed, services requiring low delay and not very high speed, etc. The existing network cannot provide high-quality of services for these scenes with huge differences in demand at the same time, but network slicing technology can well solve this problem. Therefore, it is necessary to develop network slicing technology in the 5G era. In the development process of network slicing technology, how to realize network slicing technology in the device is one of the problems that need to be solved urgently. A network slice can provide a complete end-to-end virtual network for a given user.

By dividing network resources into multiple network slices, 5G network slicing can provide differentiated services for industries with different service requirements (such as delay, reliability, capacity, isolation and other functions). The operator network will not only serve the information consumption business characterized by "best effort transmission", but also can serve the production control service with "deterministic transmission" as the communication demand. The network can allocate logically or physically isolated network resources for these services with different communication requirements. These network resources include not only the network functions of 5G core network control plane and user plane, but also include the resources of wireless network and transmission network.

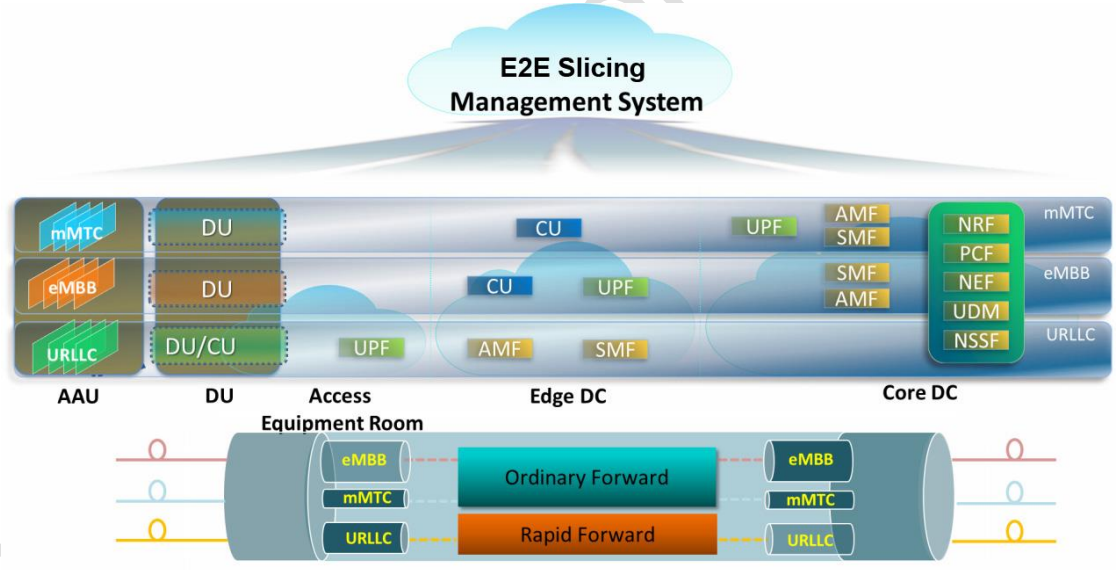


Figure 2-1 The overall framework of 5G Network Slicing

In addition, the management and opennes strategies of network operators for communication networks can be different according to the slices. The providers and users of cloud services/cloud applications can choose network management strategies of network slicing, which is suitable for their own needs, to deploy "services". Through network slicing, operators can modularize the design of network functions to meet different communication service requirements.

To sum up, both network slicing and QoS can provide different service experience for users of network communication services, but they have different connotations when providing differentiated service experience. In terms of providing differentiated user experience, the advantages of network slicing are more

prominent. Supporting multiple network slices on the device side is an important technical means for 5G devices to provide differentiated "services" for end users.

3 KEY TECHNOLOGIES AND END-TO-END PROCEDURE OF 5G NETWORK SLICING

5G devices are platforms supporting a bunch of network applications. The network applications (NetApp) are potential users of 5G network slices.

From NetApp's perspective using network slice, 5G devices are responsible for managing slice configuration information, selecting network slices for NetApp, initiating the setup of 5G PDU sessions in network slices, and maintaining these PDU sessions during mobility and networks interworking.

From 5G network's perspective, 5G devices assist the network to select network resources of transport network, access network and core network, eventually providing end-to-end slicing services over 5G network.

3.1 Network Slicing Parameters

S-NSSAI (Single Network Slice Selection Information) is used to identify a 5G network slice. The S-NSSAI represents the network resources of transport network, access network and core network used for a slice. Several S-NSSAIs form a group, called NSSAI (Network Slice Selection Information). 3GPP defined the following NSSAI to describe, maintain and manage slices. On network side, in the subscriber database, there are S-NSSAIs the user is subscribed to and related information. Such S-NSSAIs are called Subscribed S-NSSAIs.

- Default Configured NSSAI
- Configured NSSAI
- Requested NSSAI
- Allowed NSSAI
- Pending NSSAI
- Rejected NSSAI

URSP is defined in 3GPP standards to describe the relationship between traffic flows and the corresponding routing. URSP contains multiple rules for traffic flows and routing, and each rule consists in the below two parts:

- Traffic Descriptor
- Route Selection Descriptor: used to describe the S-NSSAI and other communication route characteristics that match the service flow description

The main Traffic Descriptors are listed as below, which are used for identifying which traffic flow shall be associated with which slice, RAT and other routing parameters:

- DNN
- IP Triples（IP address or IP Prefix, Port Number, Protocol ID）
- Destination FQDN
- OSId and OSAppID

3.2 Fundamental Functional Requirements for Network Slicing

3.2.1 Managing network slice parameters on device side

Devices are responsible for managing and maintaining parameters that are needed for network slices.

Devices may configure a default configured NSSAI in advance, or have a Configured NSSAI for each PLMN. The Configured NSSAI for each PLMN can also be provisioned by networks.

Before setting up a network slice for a traffic flow, a device should request for an NSSAI that can be used under the current PLMN and current registration area. To achieve this, the device should construct a Requested NSSAI based on its existing Configured NSSAI and Allowed NSSAI (if the device has saved a previous Allowed NSSAI before this procedure). After receiving the Requested NSSAI, the network will determine the set of network slices in which to register the UE and return it to the device as Allowed NSSAI.

If some S-NSSAIs in the Allowed NSSAI need authentication and authorization before they can be used, then the network will inform the device that the S-NSSAIs should be kept in Pending NSSAI before the authentication and authorization procedure is completed.

If some S-NSSAIs are not supported by the network or cannot be used temporarily, then the network will inform the device that the S-NSSAIs are kept in Rejected NSSAI. When applying a network slice for its application service flow, the device should not use the S-NSSAIs in Rejected NSSAI.

To match an application service flow with a network slice, devices use URSP that are either locally configured or configured by the network.

The network can use standard UCU (UE configuration Update) procedure to update URSP and NSSAI in the UE.

3.2.2 Setting up PDU sessions over a network slice

When a network application service flow needs communication service, the device will initiate the PDU session setup procedure. According to URSP, the device can indicate its desired S-NSSAI during the PDU session setup procedure, or the network can assign a network slice to be used for the PDU session.

Generally, if a network application service flow needs to use a dedicated network slice, then the operator should configure a corresponding network slice for the service flow in URSP. When setting up a PDU session, the device should carry the corresponding S-NSSAI to request the network to select the related slice.

3.2.3 Maintaining PDU sessions over a network slice

Network slicing is managed by each PLMN separately. When a device is roaming between 5G networks, it can provide a slice identifier that is mapped to a slice identifier in HPLMN, assisting the roaming network to set up a proper network slice for its PDU session.

For 4G/5G multimode devices, since the network slice identified by S-NSSAI is in 5GS network scope, if there is interoperability between 4G/5G networks, then the condition is relatively complex, and there needs a lot of coordination and control work between the core networks.

If the core network of 4G EPS is a dedicated core network, then when a multimode device sets up a PDN connection in the 4G network, it can assign a PDU Session ID for the PDN connection and inform the network. Based on this, the network can assign an S-NSSAI for the PDN, and send it to the device in PCO.

In idle mode, if the device reselects from EPS to 5GS, then it should send Requested NSSAI containing the previous S-NSSAI in its registration message. Besides the previous S-NSSAI, Requested NSSAI may also contain other S-NSSAIs considering Configured NSSAI and Allowed NSSAI.

In connected mode, if the device handover from EPS to 5GS, then the device obtains an Allowed NSSAI from the network side.

3.3 Protocol Signaling Procedure

3.3.1 Registration

Besides the standard S-NSSAIs defined by 3GPP, each MNO can customize their S-NSSAIs and their correspondence to SLAs. That is, within a PLMN, there is a mapping between a slice identified by an S-NSSAI and its SLA.

During the registration procedure, a device shall carry Requested NSSAI in its AS signalling to help the RAN route the registration request to the AMF, and carry Requested NSSAI in NAS signalling to notify AMF the slices it needs to use (see 3GPP TS 23.501 clause 5.15.5.2.1). AMF determines the Allowed NSSAI that a device can use by querying NSSF and UDM, and then notify it to the device in the Registration Accept message. Figure 3-1 depicts key registration messages related to network slicing.

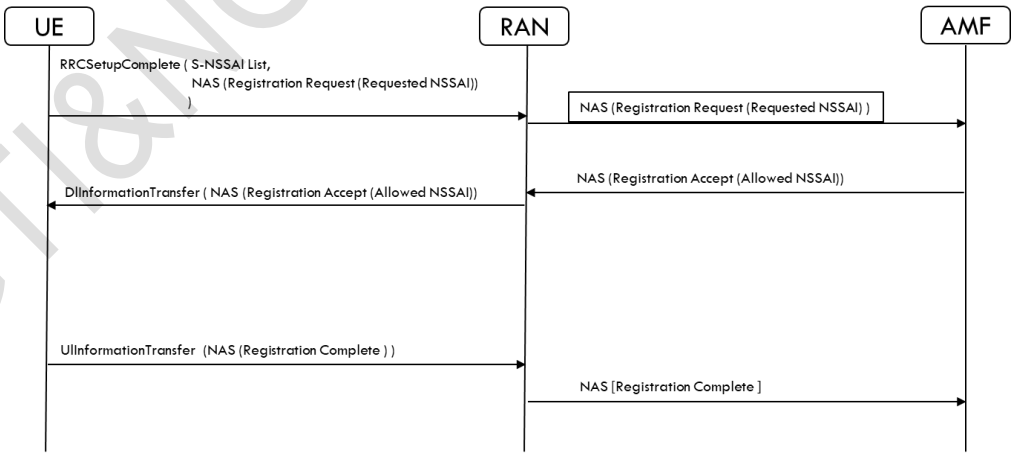


Figure 3-1 5G Device Registration for Slice Parameters

3.3.2 PDU Session Establishment

Each PDU Session the UE establishes is associated to one slice. The UE indicates which S-NSSAI, among those of the Allowed NSSAI, should be used for this PDU Session and this information is used to select the SMF and UPF.

Figure 3-2 depicts the procedures to set up a PDU session over slice.

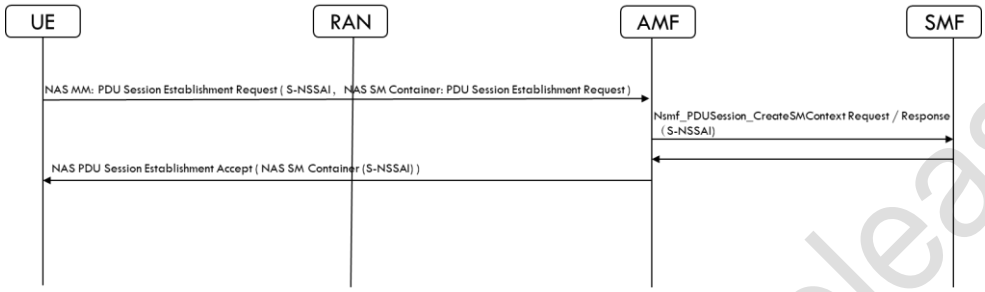


Figure 3-2 PDU session establishment

3.3.3 UE Configuration Update

OTA method can be used to update the NSSAI and URSP configuration parameters in 5G Devices.

Another method is the UCU (UE Configuration Update) procedure, which is defined in 3GPP standards to update access and mobility management parameters. According to the UCU procedure, AMF sends UE Configuration Command to UE to update/re-configure Allowed NSSAI, mapping of Allowed NSSAI, Configured NSSAI and mapping of Configured NSSAI. According to whether the updated slice configuration has an impact on the current PDU session, AMF decides whether to trigger registration messages after the procedure to complete the NSSAI update.

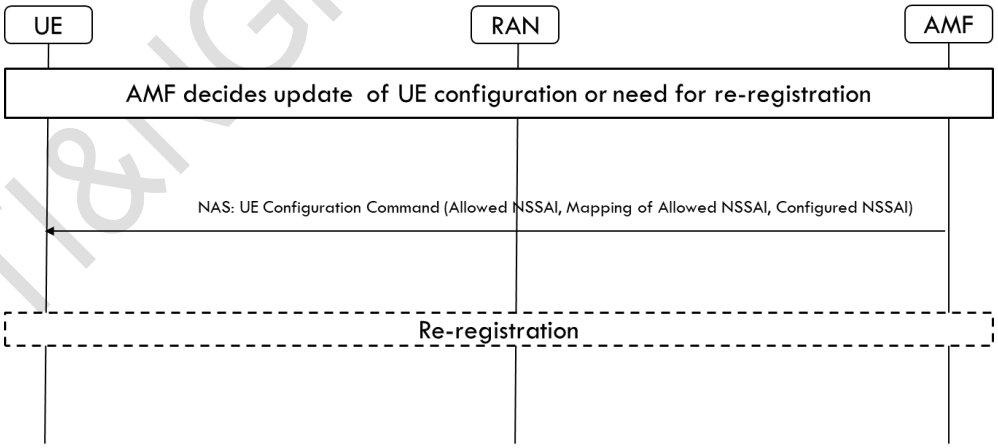


Figure 3-3 NSSAI Configuration update

URSP can be configured on devices or by PCF. When configured by PCF, the URSP is provided from PCF to AMF, and then sent to devices by AMF using UCU procedure [3GPP TS23.502\4.2.4.3]. If the device is in CM-

IDLE state, then AMF will page the device to initiate Service Request procedure. If the device is in CM-Connected state, AMF will send a UE Policy container containing URSP to the device transparently.

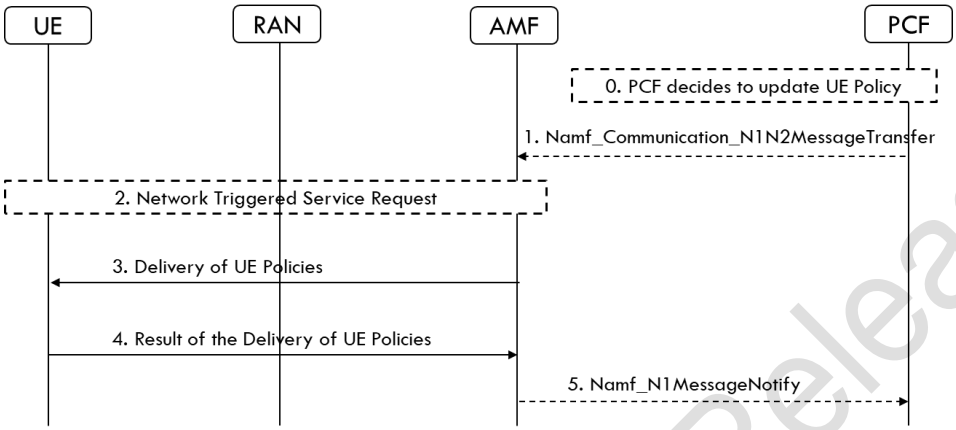


Figure 3-4 URSP update

4 CHALLENGES OF THE IMPLEMENTATION FOR 5G NETWORK SLICING IN DEVICE

4.1 Operating System not natively supporting URSP

A URSP rule is composed by a traffic descriptor (TD) and a set of route selection descriptors (RSD). The upper layer (e.g., an application) specifies the TD and the modem uses the Traffic Descriptor to look for a URSP rule matched to the TD. Given a matched Traffic Descriptor, the modem tries to establish a PDU session using the corresponding RSDs in the order of precedence. Because OS designs data connection framework based on APN type, the OS must be modified for the reason explained below to use Traffic Descriptors.

In order to hide details of data connection management and maintenance from applications, native OS characterizes a data connection by Network capability. Each Network Capability stands for a certain kind of capability. Since OS manages data connections based on APN/DNN, the capabilities associated to individual services provided by APN/DNN are the most important. Inside the OS, it is a one-to-one mapping from these capabilities to APN types, as shown in Figure 4-1, which in turn are used to build APN contexts including the DNN/APN string. Finally, the DNN/APN string is included in the parameters for the RIL to request modem to establish a PDU Session. Instead of indicating parameters for modem to make a PDU session, application can only indicate the service capability it requires, like Internet or MMS. As consequence, current native OS does not allow an application to designate directly the required traffic descriptors (App ID, DNN, IP-3tuple, FQDN, etc.)

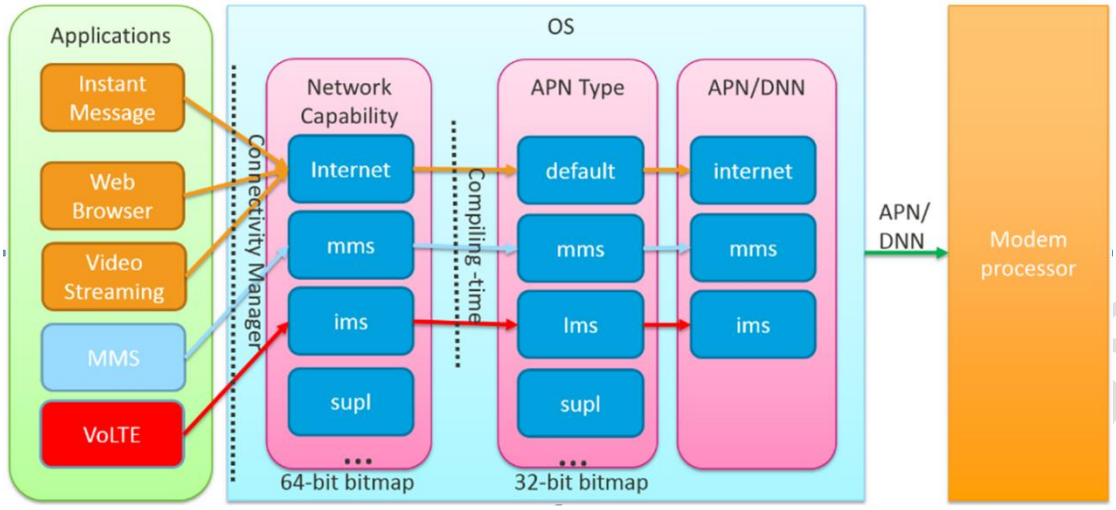


Figure 4-1: **Block diagram for Android data connection management**

Comparing different alternatives for implementing URSP related operations, extending Network Capability and support DNN based Traffic Descriptor only can be achieved with minimum design efforts. In the meantime, this minimizes the risk for OS future evolution thanks to the minimum modification to OS. Although other types of Traffic Descriptor provide more finely granularities and more flexibilities for network slice selection, more modification to OS are required. As a result, in early phase deployment for application trials, it is suggested to use DNN based Traffic Descriptor to select a network slice. At the same time, the industrial partners can devote more time to investigate solutions and unifies a solution to support all types of traffic descriptors.

4.2 Potential Modification of the Applications

In current design, Apps could ask for Internet Capability to make a network request. Moreover, some Apps do even not invoke Request Network API but create a socket directly over an active Internet PDU session. With minimal modification to OS by extending Network Capability to support DNN based Traffic descriptor, it is very likely Apps have to be modified and updated to use the new capability and utilize URSP. One more thing, since native OS does not support URSP yet, there is no single organization to manage the mapping between the new (Network Capability, APN type) pairs. This can create OS fragment issues and result in difficulties for Apps to support different platforms

4.3 Acquisition and Use of Traffic Descriptor

4.3.1 Application ID

An application ID identifies a device application. It can be used to provide precise network resource assurance for a given service using network slicing, when this service is associated with a device application (or set of applications).

Slice and service matching based on application ID presents a number of challenges. The long-standing application IDs used in different industry ecosystems, such as Android and iOS, have inconsistent format and naming rules. Therefore a given application developed for different Operating Systems and available in different application stores typically has different Application ID depending on the OS and application store. Therefore

relying on Application ID to map such application to a slice with URSP requires to know all the possible Application IDs that this application is using. In addition, there is no guarantee that no other application will use the same Application ID since there is no repository for these identities, and no verification by the device that these identities are used by the "legitimate" application. In addition, application IDs are service attributes, which are managed by the operating system at the upper layer but are hard to obtain for the modem at the bottom layer. As a result, the modem cannot establish slice connections using application IDs. In the above, "Application ID" refers to any of the two traffic descriptors specified in Table 5.2.1 of 3GPP TS 24.526: "OS Id + OS App Id type" or "OS App Id type".

4.3.2 FQDN

An FQDN is the network domain name of the target server requested by a device application. It can be used to provide network resource assurance for services in network slicing, when these services are associated with a FQDN or sets of FQDN.

Slice and service matching based on FQDN presents multiple challenges. Since FQDNs are managed by the operating system, the modem does not get the FQDN of service flow but only the IP address. It could deduce it by observing the DNS queries, but this is a complex task and this would not work if the application resolves domain names using encrypted DNS or non-DNS resolution (e.g. when CDN is deployed, the application DNS may obtain and resolve domain names through the HTTP GET method at the application layer). As a result, the modem cannot detect FQDNs of service flows, therefore cannot establish slice connections using FQDNs. In addition, a given service may involve communication with multiple domain names, some of which being used also by different services, so there is not a direct mapping between a service and a set of FQDNs.

4.3.3 Destination IP 3 Tuple

A destination IP 3 tuple is the IP address information of the target server requested by a device application. It can be used to provide network resource assurance for services in network slicing, when these services are associated with a (set of) IP address range(s).

Slice and service matching based on IP 3 tuple poses several challenges. For large-scale application services, service providers will deploy multiple servers or IP addresses. A large number of destination IP addresses leads to a long slice configuration list, making IP-address-based slice configuration difficult. In addition, destination IP addresses may change during service provisioning, which may cause failed slice configuration. Also, in configurations with multiple servers hosted behind a reverse proxy, the rules may not be sharp enough.

4.3.4 DNN

A DNN is the name of a data network used by a device application. It can be used to provide network resource assurance for services in network slicing, under the condition that a specific DNN is associated with each slice.

Slice and service matching based on DNN presents the following challenge. The native operating system abstracts a data connection as a network capability and an APN (or DNN) type, but cannot directly associate DNNs with data services. An application program implements mapping of specific data services by using a predefined limited quantity of network capabilities (such as Internet or MMS). This restricts mobile operators from flexibly configuring dedicated DNNs for specific applications. Therefore, relying in DNN as a traffic descriptor in URSP rules does not provide the flexibility and dynamicity that some use cases require.

5 REFERENCE SOLUTION OF NETWORK SLICING IN THE DEVICE

In order to provide flexible and differential services, network slicing design and provide multiple Traffic Descriptors, which can describe almost all types of business. Therefore, service providers can provide suitable network slicing service based on users' requirements and scenarios.

As mentioned above in Section 4.3, those Traffic Descriptors mainly includes as below:

- APPID
- FQDN

- Destination IP 3 Tuple
- DNN
- ...

The above Traffic Descriptors can be divided into endogenous and exogenous attributes:

- Endogenous attributes are the characteristic information that comes with the service application itself, such as APPID, FQDN and Destination IP 3 Tuple. An endogenous attribute of a service application can be one of the above attributes or the combination of the above attributes.
- Exogenous attributes are the characteristic information externally assigned to the business applicaiton, such as DNN, etc. An exogenous attribute of a service application can be one of the above attributes or the combination of the above attributes.
- Exogenous attributes can be obtained after OS starts relevant service applications and therefore OS needs to open overwrite permissions so that exogenous Traffic Descriptors can be written into the relevant service applications.

In order to realize the match between the Traffic Descriptors above and the network slicing, there are one-to-one correspondence, one-to-many correspondence and many-to-one correspondence between Traffic Descriptors and S-NSSAI. In many-to-one correspondence, for all the service applications which share the same S-NSSAI, all these service applications can share the same quality of service or different quality of services.

The realization of the slicing feature requires the addition of a slice-related SDK or software middleware in the device. According to the difference of its specific implementation methods, there are mainly two solutions:

Solution 1: Modem-Centric Solution: The matching process between service attributes and network slices is realized in the modem according to URSP.

The operating system of the device obtains the service attributes and transfers these characteristic attributes to the modem of the device, and then the modem binds the service attributes with the network slice according to the URSP. The operating system can send AT commands to the modem, and the modem extracts characteristic attributes from the AT commands.

The implementation framework of the "Modem-Centric" solution is shown in Figure 5-1:

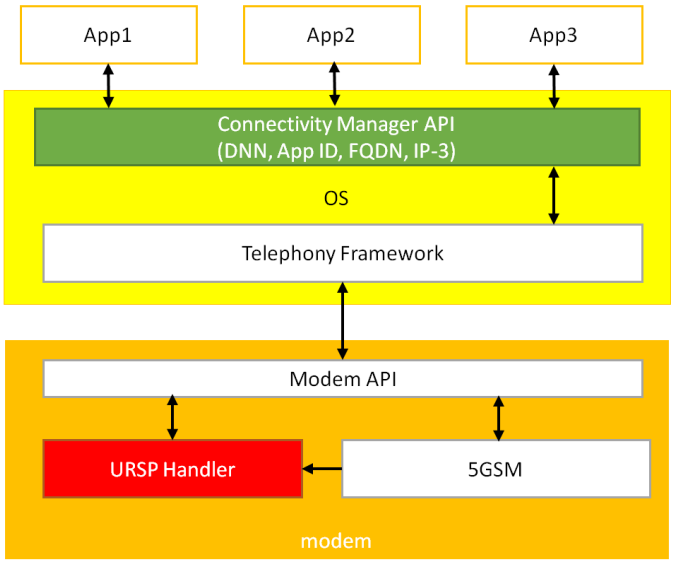


Figure 5-1 Modem-Centric reference framework

Solution 2: Operating System-Centric Solution: the matching process between service attributes and network slices is realized by the device operating system according to URSP.

In view of the difficulties and challenges in obtaining and using the Traffic Descriptor information in the URSP, it is also an ideal design to realize the acquisition and processing of URSP and TD information in the operating system on the AP side. The operating system of the device obtains the attribute information of the relevant service application, and the operating system binds the attribute information of the relevant service application with the network slice according to the URSP.

The implementation framework of OS-Centric is shown in Figure 5-2：

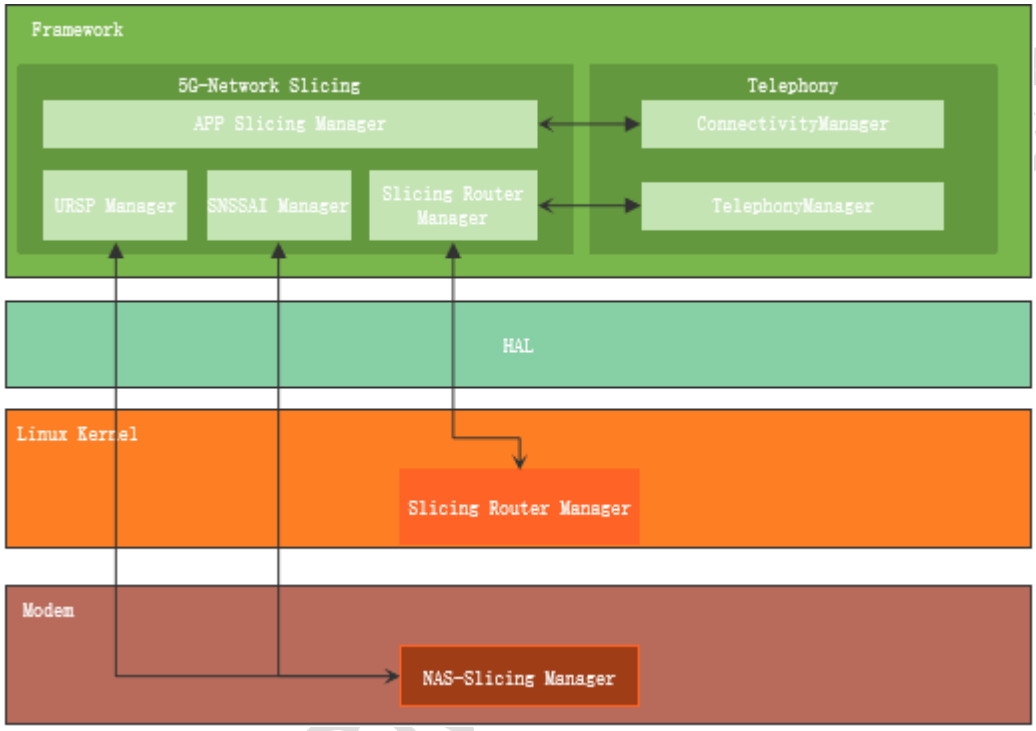


Figure 5-2 OS-Centric reference framework

In the operating system, it mainly implements the management of the slice configuration rule URSP, the configuration and management of NSSAI information, and the management and routing control of the connection between the proposed service and the slice. The Modem reports the Allowed NSSAI and URSP reported by the network side to the AP side through AT commands. The AP side performs URSP matching and routing based on the collected Traffic Descriptor (APP ID\FQDN\IP triplet\DNN) information. After the matching is successful, the device initiates the establishment of a new slice PDU session, and binds the matched service flow to the corresponding slice.

5.1 Network Slicing Solution based on APP ID

An App ID identifies a device application. It is widely recognized in the industry that the package name of a service application can be considered as an App ID. After a service application of a device is started, the device OS identifies the running status of the application in the internal system environment, and obtains the App ID (package name) of the service application.

After determining an App ID, the device compares and matches the ID with the URSP rules. If there is a matching rule, it binds the ID with the corresponding S-NSSAI in the rule.If there is no matching rule in the non-default URSP, it follows the behavior described in 3GPP TS 24.526".

5.2 Network Slicing Solution based on FQDN

FQDN is the network domain name of the target server requested by a device service application. The Netd module in the OS resolves the FQDN of a service application. Therefore, when a service application initiates an FQDN resolution request, the device OS can obtain the FQDN of the service application.

After determining an FQDN, the device compares and matches the FQDN with the URSP rules. If there is a matching rule, it binds the FQDN with the corresponding S-NSSAI in the rule.If there is no matching rule in the non-default URSP, it follows the behavior described in 3GPP TS 24.526".

5.3 Network Slicing Solution based on IP 3 Tuples

A destination IP 3 tuple is the IP address information of the target server requested by a device service application. The kernel in the OS processes data packets sent by service applications. Therefore, the device OS can obtain destination IP 3 tuples of data packets.

After determining a destination IP 3 tuple, the device compares and matches the tuple with the URSP rules. If there is a matching rule, it binds the tuple with the corresponding S-NSSAI in the rule. If there is no matching rule in the non-default URSP, it follows the behavior described in 3GPP TS 24.526".

5.4 Network Slicing Solution based on DNN

5.4.1 DNN-Based Solution 1

Solution 1 is quite straightforward. In the data connection framework, additional (Network Capability, APN type) pairs are added, new pair for each DNN associated to a network slice (can be the same slice). An App can request a network service with the newly defined NetworkCapability, e.g., NET_Capbility_DNN1 or NET_Capbility_DNN2.

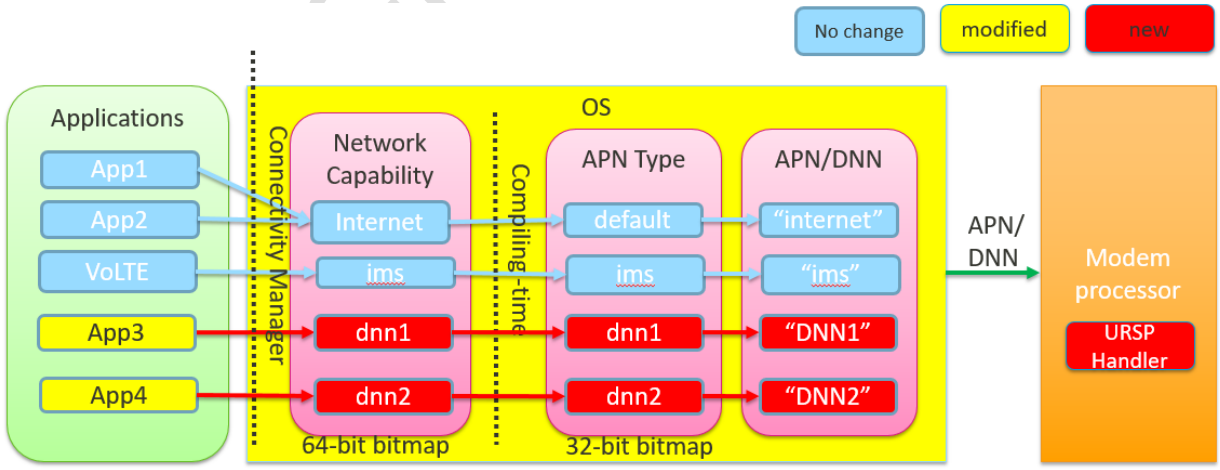
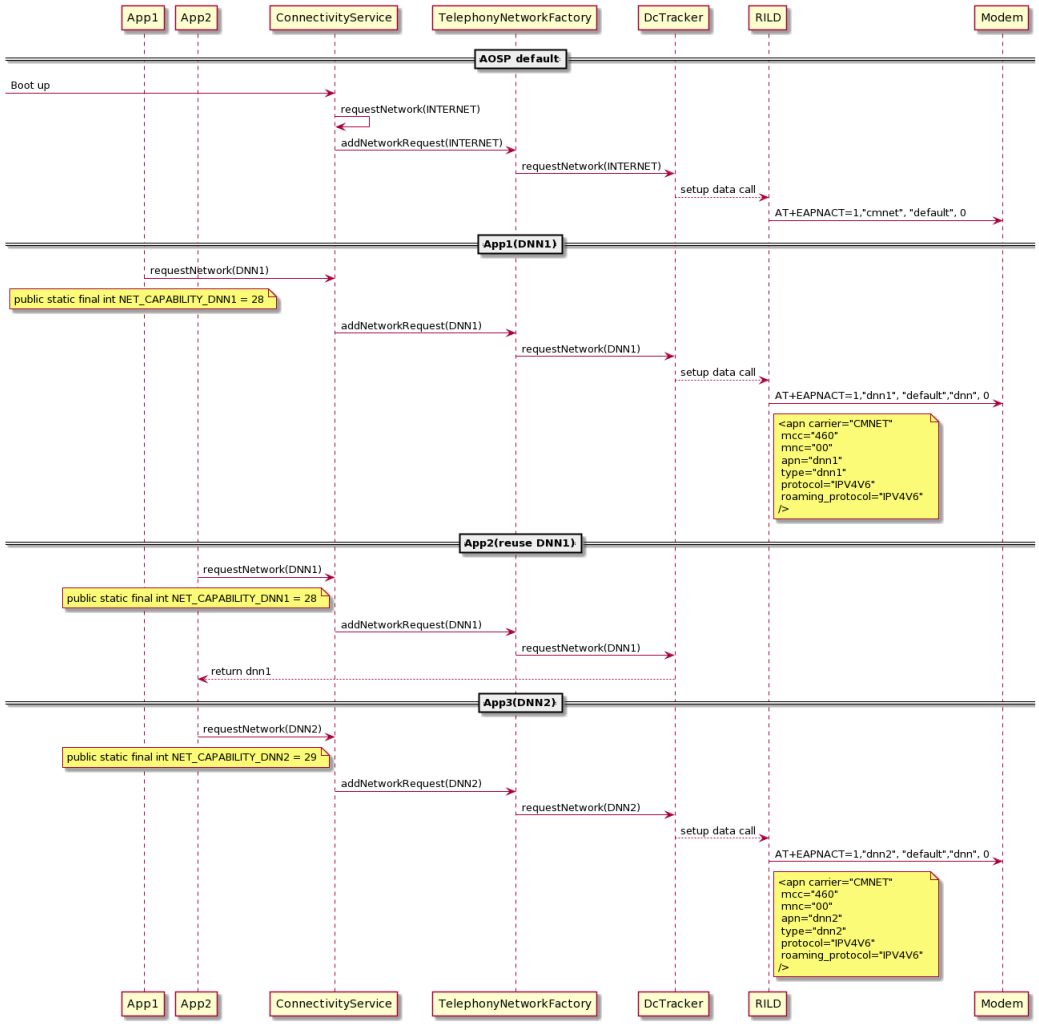


Figure 5-3: DNN-Based Solution 1 Block Diagram

Below illustrate the detailed procedures.



5.4.2 DNN-Based Solution 2

In order to allow more DNN can be used, class NetworkSpecifier is extended to include DNN string such that an App can specify the desired DNN when making a network request as shown in **错误!未找到引用源。**.

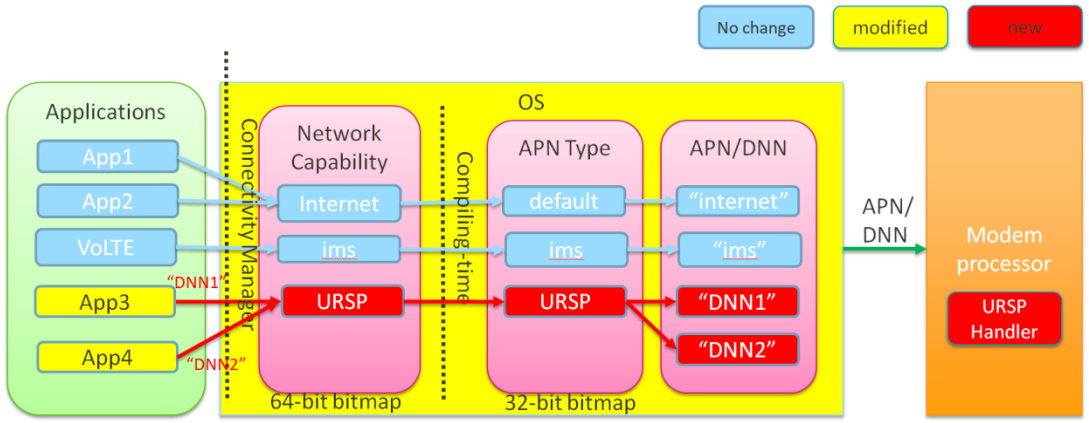


Figure 5-4: DNN-Based Solution 2 Block Diagram

5.4.3 Summary and Comparison for DNN-based Solutions

The table below lists a comparison between solution 1 and 2.

Solution	Brief Description	Pros and Cons
Solution 1	1. Add additional (NetworkCapability, APN type) pairs 2. Application request the new NetworkCapability assigned based on subscription.	1. Minimal change to OS， less migration risk 2. Limited number of (NetworkCapability, APN type) pairs available for extension. 3. Possible conflicts if later OS releases define new NetworkCapability.
Solution 2	1. Add one new (NetworkCapability, APN type) pair 2. Extend NetworkSpecifier to include DNN string 3. Apps use the new capability and include DNN string to request network service	1. No limitation to number of DNNs 2. More changes to OS framework and higher OS migration risk.

5.5 Network Slicing Solution based on Multiple Traffic Descriptors

When there are multiple traffic descriptors, the AND logic is used for rule matching. When all traffic descriptors of the URSP are matched successfully, it binds the ID with the corresponding S-NSSAI in the rule. If there is no matching rule in the non-default URSP, it follows the behavior described in 3GPP TS 24.526". The overall process shown in Figure 5-5

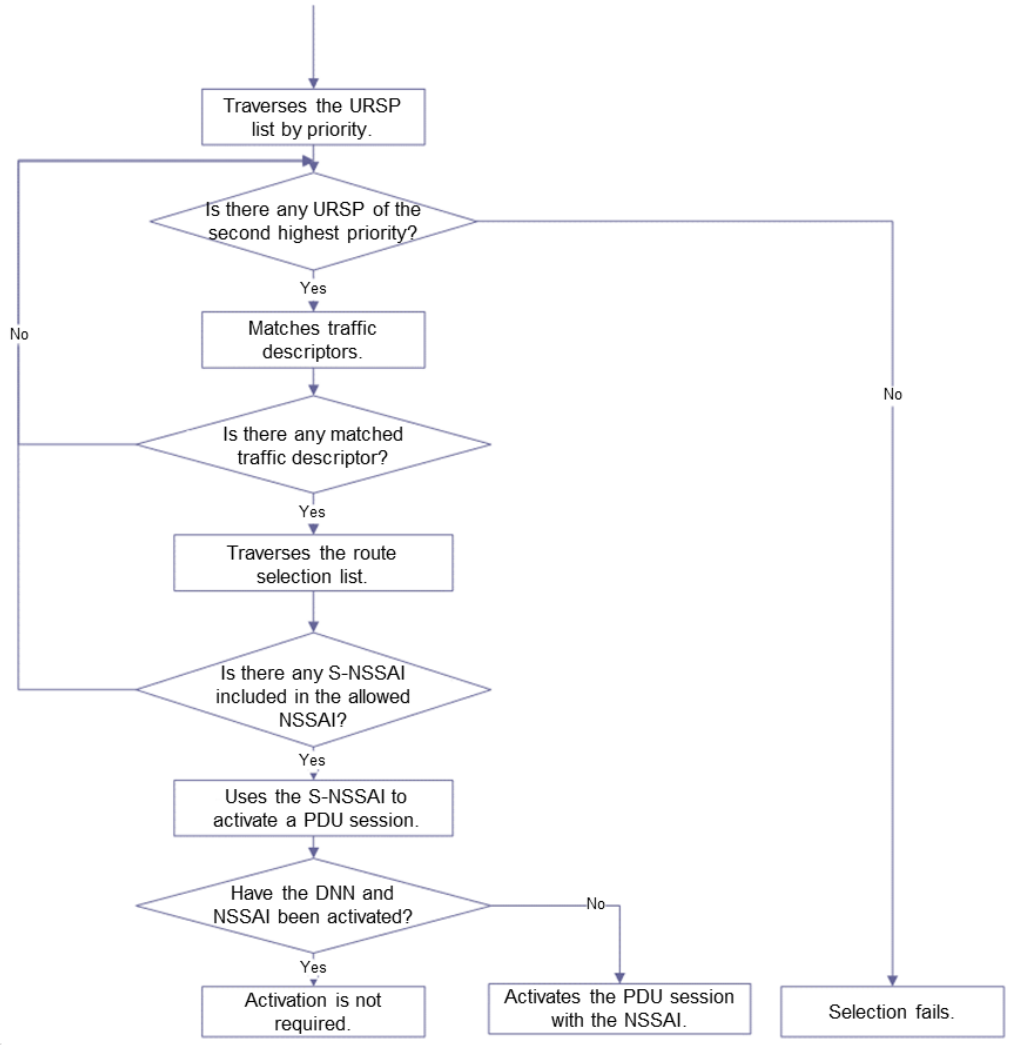


Figure 5-5: Multiple Traffic Descriptors Matching Process

6 TEST REQUIREMENTS FOR 5G NETWORK SLICING IN DEVICE

6.1 Testing requirements.

A key principle of the device testing requirements is to ensure 'inter-operability' of devices and networks, This is to ensure that a user has the expected level of functionality and user experience as they move between different networks, and they do not have to take care on which network they use and how that network is configured to behave. This is also to ensure that a network does not suffer any degradation of performance issues when devices operate on the network, and that the network provides stable operation regardless of which devices are connected to the network.

The principle testing requirements are broken down into two categories; 'standards compliance' for the exchange of messages (i.e. NSSAI and URSP configuration information) between a network and the device, and 'UE implementation' for the behaviour of the device in using the URSP for selecting and requesting certain slice configurations. The 'standards compliance' is normally contained in the 3GPP RAN5 testing requirements and methodology, and supported by testing schemes such as GCF, PTCRB, GTI, etc. The 'UE implementation' is normally an Operator or device vendor specific test plan. This 'UE implementation' testing should test the way in which the application/device uses the configuration information, and the triggers which cause the device to request/change the network slice and routing that is being used. These aspects are designated as "UE implementation specific" and do not have a standardised behaviour or test procedure.

To enable testing at an 'end to end' level requires the inclusion of application layer functions at the device side, as the slice selection and traffic routing procedures use application related selection criteria and mappings that are configured within the device. This application layer and device implementation specific functionality is currently outside of the scope being developed by 3GPP RAN5. Suitable test interfaces and test/verification procedures to support the implementation and inter-operability / consistency of Network Slicing for Smartphones are required.

6.2 Test coverage and existing capabilities.

3GPP has specified conformance requirements for NSSAI and URSP, reference TS31.124 (for USAT) and TS38.523-1 (for Protocol). These cover the exchange of configuration information between the network and device.

GCF has the following 'standards compliance' related work items:

- WI-509-NR USAT test cases which cover the refresh of USAT information via NAS messages (e.g. URSP) and the set-up of a new bearer session after new URSP configuration messages. Ref TS31.124.
- WI-504-NR Protocol test cases which cover the handling and storage of NSSAI configuration messages, exchange of Network Slice information during registration procedures, and use of Network Slice information during PDU session procedures. Ref TS38.523-1.

6.3 Network Slicing test areas.

Exchange of NSSAI information between Network and UE (reading, authorising and updating available slices), is covered by 3GPP Conformance Tests in TS38.523-1. This verifies the transfer of required information across the air interface signalling.

Priority rules for network informed, and UE configured (including any default) slice preferences and options are present in 3GPP standards TS23.501 section 5.15.4. These must be validated to ensure any configurations provisioned by third party are correctly handled versus network provided configurations. This is a UE implementation specific area, outside of current 3GPP conformance test coverage. The text in 3GPP TS23.501

section 5.15.5 provides the detailed information for handling NSSAI. This includes the implementation of NSSAI rules, procedures to register and modify the NSSAI lists in the UE, and process to establish a PDU session using these lists.

6.4 URSP test areas.

USIM based refresh of Routing Indicator data via NAS messages, and opening a new radio bearer after receiving URSP policy update from the network is covered by 3GPP in TS 31.124. This verifies the transfer of required information across the air interface signalling to the USIM, for any USIM stored/configured URSP rules.

3GPP TS 23.503 section 6.6.2.3 provides the UE procedure for associating applications to PDU sessions based on URSP information. The behaviour of the UE to implement these procedures is the area of testing that is 'UE implementation specific'. Mapping of applications to URSP rules, and the selection process for different URSP rules, is UE implementation specific and outside of current 3GPP conformance test coverage. If Operators are providing specific URSP rules with required parameters/settings, then the mapping in the UE should be checked for consistency in selection and prioritisation of rules and mapping to PDU session types. UE may also have 'Local Configurations' which are used for selection of a specific PDU session type for a specific application (stored in the UE or in the USIM). Equivalently, if the same rules are provisioned to different UE's, then the interpretation and response to these rules by each UE should be tested for consistency and correct expect behaviour. In the case of roaming, there may be differences in rules provided by the H-PLMN and the V-PLMN, and the correct priority handling of these rules should be verified.

3GPP TS 24.526 describes the UE Policies for URSP. Section 4.2 covers the process of handling URSP, and section 5 covers the details of encoding the URSP rules. URSP rules can be pre-configured in the USIM and/or UE, and can also be provided by network operator (from the PCF) using the NAS messaging 'UE Policy Delivery Service'. Correct provisioning and inter-operability (equivalence) of rules may need to be verified to ensure consistent/predictable selection behaviour by UE's when operating in different networks (e.g. roaming scenarios).

There is no standardised way to trigger URSP rule selection, driven by application layer and is UE implementation specific. To enable testing then the UE must be provided with a suitable test method (e.g. remote command) or a suitable test application that can trigger the required selection procedures.

6.5 Application layer and Operating System

For the application layer, the presence of suitable 3GPP specifications for the interface between Application Server and 3GPP network system provide a basis for inter-operability testing. Further analysis of GTI inter-operability requirements to identify suitable inter-operability test environments may be needed, based on SEAL reference architecture, and then mapped onto relevant interface specifications.

The process of matching UE application (PDU session) to URSP and requesting suitable network slices is covered in section 3 of this document, and different options are referred to in section 5 of this document. Each of the different methods referenced in section 5 require a combination of NSSAI and URSP provisioning, followed by Application specific selection of rules/profiles which are a UE/OS specific implementation. To verify the correct operation requires a combination of the NSSAI tests, URSP tests, and then the UE/OS selection procedures. To trigger the required selections of specific profiles will require either a 'test application' or specific test interfaces in the OS, that enables a specific selection in the UE.

6.6 Testing methodology

6.6.1 NSSAI tests

The NSSAI provisioning procedures are covered in 3GPP RAN5 Protocol test cases.

The priority selection of different NSSAI's is not covered in 3GPP RAN5 (as it is a UE implementation specific feature) but testing of the UE compliance to the prioritisations rules is required for stable and predictable Network Slicing service. So, a specific test suite is required to verify the priority of different priority handling between USIM provisioned, UE provisioned, and Network provisioned NSSAI's.

6.6.2 URSP tests

The provisioning of URSP rules from network (PCF) to UE over the NAS 'UE Policy Delivery Service' messaging should be verified to ensure correct provisioning of the rules. This testing is not covered in 3GPP RAN5 Protocol test cases. This should be an operator specific test case. The UE may also be provisioned with URSP rules from the USIM or UE vendor, so a specific test suite is required to verify the priority of different rule handling between USIM provisioned, UE provisioned, and Network provisioned (both H-PLMN and V-PLMN) URSP rules.

6.6.3 'End to end' tests

The mapping of Applications to URSP rules and NSSAI's, and therefore mapping Applications to PDU sessions, is a UE implementation specific feature. To test this, firstly a set of reference NSSAI configurations and URSP Rules should be correctly provisioned into the UE (see sections above). Secondly, the UE/Application should be triggered to select an existing or new PDU session, based on the provisioned rules, and then a corresponding data session should be established using either a specific current or specific new PDU session. Based on the provisioned rules, the test should verify that the correct PDU session is used for the specific configuration of the application.

The processing of rules and selection of PDU session can be based on several different parameters/methods, as described in section 5. Each type of selection process should be tested with corresponding URSP rules.

7 SUMMARY

7.1 Summary of 5G network slicing scheme in the device

Generally speaking, 5G network slicing scheme in the device can be divided into two categories:

- The basic scheme of 5G network slicing in the device: It has the basic network slicing service capability based on APPID, FQDN, IP3 tuple, DNN and other traffic granularity;
- The target scheme of 5G network slicing in the device: Based on the basic scheme of 5G network slicing in the device, the customized DNN is added.The target scheme provides the enhance service capability based on APPID, FQDN, IP3 tuple, customized DNN and other traffic granularity.

The basic scheme of 5G network slicing in the device can provide operators and users with network slicing services based on APPID, FQDN, IP3 tuple, DNN and other service granularity, which has good scenario applicability.

APP ID can well solve the problems such as the uniqueness of application identity and security of transmission, which is particularly important for diversified development and reliable operation of network slicing in the smartphone. This scheme requires the operating system to support the acquisition of traffic APP's characteristic attributes (TD), and requires a certain amount of operating system renovation work. At present, network slicing solutions have been developed independently by device vendors. Those solutions support APP ID, FQDN, IP 3 tuple, DNN and other full strategies based on the existing operating system

architecture through deep adaptation of chips, to meet the flexible and diverse demands of various application scenarios for network slicing.

In addition, although the basic scheme can support DNN, these DNNs are limited to Internet, IMS and MMS, and do not support the customized DNN. Based on the basic scheme of network slicing, the target scheme of network slicing in the device improves the information transmission ability of DNN, realizes the flexible transmission ability of customized DNN parameters, and enhances the connection between operators and applications. However, the customized DNN transmission scheme needs to expand the network capability/APN type parameter or modify the existing OS data connection establishment process, and the compatibility and forward usability of future OS versions need to be fully considered. Furthermore, DNN, as an external characteristic parameter, has the risk of being stolen and falsely used, which is also a problem to be solved in the future.

From the point of view of meeting the diversification of user requirements to the greatest extent, 5G device should support the target scheme of network slicing, i.e. it should be able to provide network slicing service based on APPID, FQDN, IP3 tuple, DNN and other traffic attribute information. At the same time, 5G device should also support customized DNN scheme, further expand the ability of operators to provide network slicing services for users, and improve the user experience.

In the design of device system architecture, network slicing features require the coordination between the upper operating system and the bottom communication modem. At present, there are two ways to implement network slicing features in the device system architecture

1. Modem centric scheme: the matching process between traffic attributes and network slices is implemented in modem;
2. OS centric scheme: the matching process between traffic attributes and network slices is implemented in the operating system.

Considering that the network slicing features are the capability characteristic of mobile network, in order to maintain the following capacity expansion, "upper-bottom" standardized adaptation, service attribute perception and other reasons, choosing "modem centralization scheme" will provide users with more diversified, flexible and evolvable high-quality network slicing services.

7.2 Prospect of network slicing in the device

At present, the mainstream operating systems, such as Android and IOS, have not yet provided network slicing solutions based on the native operating systems. Therefore, some industry manufacturers are exploring the design and development of their own network slicing solutions in the device. These schemes are different in route selection and technical implementation methods. Some schemes focus on the implementation of upper OS, while others focus on the modem baseband. Different design schemes have different logic and process, so we have to consider whether the logic of OS and chip manufacturers will have conflict or not. How to modify the native framework of OS is a problem that device manufacturers must face in the transformation of operating system.

Moreover, the service capability of network slicing is different, which also makes the potential risk that the network slicing service will be embezzled and falsely used by malicious parties. 3GPP provides secondary authentication and authorization (R15 feature) and network specific authentication and authorization (R16 feature), non-3GPP identity or certificate can be used.

Table 7-1: Comparison of 3GPP authentication / authorization mechanisms

	Main Authentication	Secondary Authentication	Network Slicing

			Authentication
Framework	5G / EAP	EAP	EAP
Algorithm	5G-AKA / EAP-AKA'	Specified by external network	Specified by external network signing by network slicing
Identity	SUCI / SUPI， store in USIM/UDM	Managed by external network, in NAI format, stored in device and AAA server	Managed by external network signing by network slicing, stored in device and AAA server
Credentials	Store in USIM/UDM	Managed by external network, stored in device and AAA server	Managed by external network signing by network slicing, stored in device and AAA server
Trigger	Network registration	PDU session establishment	Network registration
Authenticator	AMF	SMF	AMF
AAA server	AUSF	Operator administer or external network	Operator administer or external network

However, the native operating system does not support secondary authentication and special authentication for network slicing, which needs to be realized jointly by device manufacturers and chip manufacturers. In the future, all parties in the industry should also further explore 3GPP or non-3GPP solutions to find more fully complete technical implementation solutions, and then carry out the industry standard.

7.3 Conclusion

The success of 5G technology lies in its ability to meet a variety of business requirements. In today's highly connected world, there are different types of devices everywhere, and the "one-fit-all" wireless network cannot meet all kinds of industrial requirements. In this regard, operators provide operators' infrastructures in a shared way through network slicing technology, create and support multiple virtual networks on the infrastructure of public network, ensure the high-quality of service requirements of various kinds of services through isolation protection, and provide differentiated services for the devices to meet the diversified requirements of applications, users, vertical industries and other fields.

5G network slicing is a technical innovation of 5G network. For operators, the large number of end users are the business basis of operators. In order to realize the application of network slicing in device, operators put forward the 5G network slicing solution in the device with industrial partners, and analyzed the difficulties and challenges faced by the implementation of 5G network slicing in the device.

The application of network slicing in device side is a new direction to develop. Using network slicing technology at the device side can ensure the high-quality experience requirements of device services through isolation for end users. For the overall development of network slicing industry, it can promote further in-depth cooperation among individual users, third-party applications, mobile phone manufacturers, chip manufacturers, operating system manufacturers and operators, and gradually promote new business models and new ecological environment.

8 APPENDIX

8.1 DNN-Based Solution 1（Reference code of DNN indirect delivery scheme）

8.1.1 Changes to OS APIs：

1) Define new network capabilities and APN types

（1） frameworks/base/core/java/android/net/NetworkCapabilities.java

```
@Retention(RetentionPolicy.SOURCE)
  @IntDef(prefix = { "NET_CAPABILITY_" }, value = { NET_CAPABILITY_MMS,
      NET_CAPABILITY_SUPL,
      NET_CAPABILITY_DUN,
      NET_CAPABILITY_FOTA,
      NET_CAPABILITY_IMS,
      NET_CAPABILITY_CBS,
      NET_CAPABILITY_WIFI_P2P,
      NET_CAPABILITY_IA,
      NET_CAPABILITY_RCS,
      NET_CAPABILITY_XCAP,
      NET_CAPABILITY_EIMS,
      NET_CAPABILITY_NOT_METERED,
      NET_CAPABILITY_INTERNET,
      NET_CAPABILITY_NOT_RESTRICTED,
      NET_CAPABILITY_TRUSTED,
      NET_CAPABILITY_NOT_VPN,
```

NET_CAPABILITY_VALIDATED,

NET_CAPABILITY_CAPTIVE_PORTAL,

NET_CAPABILITY_NOT_ROAMING,

NET_CAPABILITY_FOREGROUND,

NET_CAPABILITY_NOT_CONGESTED,

NET_CAPABILITY_NOT_SUSPENDED,

NET_CAPABILITY_OEM_PAID,

NET_CAPABILITY_MCX,

NET_CAPABILITY_PARTIAL_CONNECTIVITY,

NET_CAPABILITY_TEMPORARILY_NOT_METERED,

NET_CAPABILITY_DNN1,

NET_CAPABILITY_DNN2, })

public static final int NET_CAPABILITY_DNN1 = 27; // Sample value

public static final int NET_CAPABILITY_DNN1 = 28; // Sample value

（2） frameworks\base\telephony\java\com\android\internal\telephony\PhoneConstants.java

public static final String APN_TYPE_DNN1 = ApnSetting.TYPE_DNN1_STRING;

public static final String APN_TYPE_DNN2 = ApnSetting.TYPE_DNN2_STRING;

public static final String[] APN_TYPES = {APN_TYPE_DEFAULT, APN_TYPE_MMS,

APN_TYPE_SUPL,

APN_TYPE_DUN,

APN_TYPE_HIPRI,

APN_TYPE_FOTA,

APN_TYPE_IMS,

APN_TYPE_CBS,

APN_TYPE_IA,

APN_TYPE_EMERGENCY,

APN_TYPE_MCX,

APN_TYPE_XCAP,

APN_TYPE_DNN1,

APN_TYPE_DNN2, };

（3） frameworks\base\telephony\java\android\telephony\data\ApnSetting.java

public static final int TYPE_DNN1 = ApnTypes.DNN1;

public static final int TYPE_DNN2 = ApnTypes.DNN2;

```
...
public static final String TYPE_DNN1_STRING = "dnn1";
public static final String TYPE_DNN2_STRING = "dnn2";
static { APN_TYPE_STRING_MAP = new ArrayMap<>();
  APN_TYPE_STRING_MAP.put(TYPE_ALL_STRING, TYPE_ALL);
  APN_TYPE_STRING_MAP.put(TYPE_DEFAULT_STRING, TYPE_DEFAULT);
  APN_TYPE_STRING_MAP.put(TYPE_MMS_STRING, TYPE_MMS);
  APN_TYPE_STRING_MAP.put(TYPE_SUPL_STRING, TYPE_SUPL);
  APN_TYPE_STRING_MAP.put(TYPE_DUN_STRING, TYPE_DUN);
  APN_TYPE_STRING_MAP.put(TYPE_HIPRI_STRING, TYPE_HIPRI);
  APN_TYPE_STRING_MAP.put(TYPE_FOTA_STRING, TYPE_FOTA);
  APN_TYPE_STRING_MAP.put(TYPE_IMS_STRING, TYPE_IMS);
  APN_TYPE_STRING_MAP.put(TYPE_CBS_STRING, TYPE_CBS);
  APN_TYPE_STRING_MAP.put(TYPE_IA_STRING, TYPE_IA);
  APN_TYPE_STRING_MAP.put(TYPE_EMERGENCY_STRING, TYPE_EMERGENCY);
  APN_TYPE_STRING_MAP.put(TYPE_MCX_STRING, TYPE_MCX);
  APN_TYPE_STRING_MAP.put(TYPE_XCAP_STRING, TYPE_XCAP);
  APN_TYPE_STRING_MAP.put(TYPE_DNN1_STRING, TYPE_DNN1);
  APN_TYPE_STRING_MAP.put(TYPE_DNN2_STRING, TYPE_DNN2);

  APN_TYPE_INT_MAP = new ArrayMap<>();
  APN_TYPE_INT_MAP.put(TYPE_DEFAULT, TYPE_DEFAULT_STRING);
  APN_TYPE_INT_MAP.put(TYPE_MMS, TYPE_MMS_STRING);
  APN_TYPE_INT_MAP.put(TYPE_SUPL, TYPE_SUPL_STRING);
  APN_TYPE_INT_MAP.put(TYPE_DUN, TYPE_DUN_STRING);
  APN_TYPE_INT_MAP.put(TYPE_HIPRI, TYPE_HIPRI_STRING);
  APN_TYPE_INT_MAP.put(TYPE_FOTA, TYPE_FOTA_STRING);
  APN_TYPE_INT_MAP.put(TYPE_IMS, TYPE_IMS_STRING);
  APN_TYPE_INT_MAP.put(TYPE_CBS, TYPE_CBS_STRING);
  APN_TYPE_INT_MAP.put(TYPE_IA, TYPE_IA_STRING);
  APN_TYPE_INT_MAP.put(TYPE_EMERGENCY, TYPE_EMERGENCY_STRING);
  APN_TYPE_INT_MAP.put(TYPE_MCX, TYPE_MCX_STRING);
  APN_TYPE_INT_MAP.put(TYPE_XCAP, TYPE_XCAP_STRING);
  APN_TYPE_INT_MAP.put(TYPE_DNN1, TYPE_DNN1_STRING);
  APN_TYPE_INT_MAP.put(TYPE_DNN2, TYPE_DNN2_STRING); ...
```

（4） frameworks\opt\telephony\src\java\com\android\internal\telephony\dataconnection\DataConnection.java

```
public NetworkCapabilities getNetworkCapabilities() {
    ...
    case PhoneConstants.APN_TYPE_XCAP: {
      result.addCapability(NetworkCapabilities.NET_CAPABILITY_XCAP);
      break;
    }
    case PhoneConstants.APN_TYPE_DNN1: {
      result.addCapability(NetworkCapabilities.NET_CAPABILITY_DNN1);
      break;
    }
    case PhoneConstants.APN_TYPE_DNN2: {
      result.addCapability(NetworkCapabilities.NET_CAPABILITY_DNN2);
      break;
    }
    default:
    ...
  }
```

（5） frameworks\opt\telephony\src\java\com\android\internal\telephony\PhoneSwitcher.java

```
protected NetworkCapabilities makeNetworkFilter() {
    NetworkCapabilities netCap = new NetworkCapabilities();
    netCap.addTransportType(TRANSPORT_CELLULAR);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_MMS);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_SUPL);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_DUN);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_FOTA);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_IMS);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_CBS);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_IA);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_RCS);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_XCAP);
```

```
netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_EIMS);
    netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_NOT_RESTRICTED);
netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET);
netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_MCX);
netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_DNN1);
netCap.addCapability(NetworkCapabilities.NET_CAPABILITY_DNN2);
netCap.setNetworkSpecifier(new MatchAllNetworkSpecifier());
    return netCap;
}
```

（6） frameworks\opt\telephony\src\java\com\android\internal\telephony\dataconnection\TelephonyNetworkFactory.java

```
protected NetworkCapabilities makeNetworkFilter(int subscriptionId) {
    NetworkCapabilities nc = new NetworkCapabilities();
    nc.addTransportType(NetworkCapabilities.TRANSPORT_CELLULAR);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_MMS);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_SUPL);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_DUN);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_FOTA);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_IMS);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_CBS);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_IA);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_RCS);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_XCAP);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_EIMS);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_NOT_RESTRICTED);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_DNN1);
    nc.addCapability(NetworkCapabilities.NET_CAPABILITY_DNN2);
    nc.setNetworkSpecifier(new TelephonyNetworkSpecifier.Builder()
.setSubscriptionId(subscriptionId).build());
    return nc;
}
```

8.1.2 Map network capability to APN type

frameworks\opt\telephony\src\java\com\android\internal\telephony\dataconnection\ApnContext.java

```
public static @ApnType int getApnTypeFromNetworkRequest(NetworkRequest nr) { ...
    if (nc.hasCapability(NetworkCapabilities.NET_CAPABILITY_XCAP)) {
      if (apnType != ApnSetting.TYPE_NONE) error = true;
      apnType = ApnSetting.TYPE_XCAP;
    }
    if (nc.hasCapability(NetworkCapabilities.NET_CAPABILITY_DNN1)) {
      if (apnType != ApnSetting.TYPE_NONE) error = true;
      apnType = ApnSetting.TYPE_DNN1;
    }
    if (nc.hasCapability(NetworkCapabilities.NET_CAPABILITY_DNN2)) {
      if (apnType != ApnSetting.TYPE_NONE) error = true;
      apnType = ApnSetting.TYPE_DNN2;
    }
    ...
  }
```

8.1.3 Include operator specified APN to apn-config.xml and assign APN type to the newly added DNN types, e.g., DNN1 or DNN2.

```
<apn carrier="CMNET"
  mcc="460"
  mnc="00"
  apn="dnn1"
  type="dnn1"
  protocol="IPV4V6"
  roaming_protocol="IPV4V6"
  />
```

```
<apn carrier="CMNET"
  mcc="460"
  mnc="00"
  apn="dnn2"
  type="dnn2"
  protocol="IPV4V6"
  roaming_protocol="IPV4V6"
/>
```

8.1.4 Changes to AOSP HIDL APN type (hardware\interfaces\radio\1.5\types.hal)

```
enum ApnTypes : @1.4::ApnTypes { /**
    * APN type for XCAP
    * NOTE: Due to the addition of this new value, the value ALL defined in
```

```
     * 1.0::ApnTypes is deprecated and should not be used.
     */ XCAP = 1 << 11,
    // reserve some bit
    DNN1 = 1 << 15, // sample value
    DNN2 = 1 << 16, // sample value
    ...
};
```

8.1.5 RILD Invokes modem functionalities to select a network slice to establish a PDU session together with other parameters in RSD

<<Depends on OEM implementation>>

8.2 DNN-Based Solution 2（Reference code of DNN direct delivery scheme）

Changes to OS APIs：

1. **Similar to solution 1, add a new Network Capability, NET_CAPABILITY_DNN, and a new APN type, APN_TYPE_DNN. Associate APN_TYPE_DNN to NET_CAPABILITY_DNN .**

2. **Inherite NetworkSpecifier and create a new class, DnnNetworkSpecifier, to support inclusion of DNN string.**

```
public final class DnnNetworkSpecifier extends NetworkSpecifier implements Parcelable {
  /**
  * Arbitrary string used to pass (additional) information to the network factory.
  */
  @UnsupportedAppUsage
  public final String mDnn;
  public final int mSubId;

  public DnnNetworkSpecifier(int subId, String dnn) {
    this.mSubId = subId
    this.mDnn = dnn;
    }
    @Override
    public boolean satisfiedBy(DnnNetworkSpecifier other) {
      return mDnn == other.mDnn;
    }
```

...

}

3. **Modify DcTracker.java and extend APN context list. For network request with APN_TYPE_DNN, create a new APN context and insert it into a newly designed APN context list (mDnnApnContext).**

```
protected final ConcurrentHashMap<String, ApnContext> mDnnApnContexts = new
ConcurrentHashMap<String, ApnContext>();
public void requestNetwork(NetworkRequest networkRequest, @RequestNetworkType int type,
                    Message onCompleteMsg) {
   final int apnType = ApnContext.getApnTypeFromNetworkRequest(networkRequest);
   if (apnType == APN_TYPE_DNN) {
      String[] networkConfigStrings = mPhone.getContext().getResources().getStringArray(
            com.android.internal.R.array.networkAttributes);
      NetworkConfig networkConfig = new NetworkConfig(networkConfigString);
      String dnn = (DnnNetworkSpecifier)(networkRequest.getNetworkSpecifier()).mDnn;
      addDnnApnContext(dnn);
   }
   final ApnContext apnContext = mApnContextsByType.get(apnType);
   if (apnContext != null) {
      apnContext.requestNetwork(networkRequest, type, onCompleteMsg);
   }
}
protected ApnContext addApnContext(String dnn, NetworkConfig networkConfig) {
   ApnContext apnContext = new ApnContext(mPhone, APN_TYPE_DNN, mLogTag, networkConfig,
this);
   mApnContexts.put(dnn, apnContext);
   return apnContext;
}
```

4. **Modify AOSP reuse logic in DcTracker.java. For APN_TYPE_DNN, need additionally check if the DNN string is the same or not before reuse a network connection in the following functions.**

```
private void onEnableApn(@ApnType int apnType, @RequestNetworkType int requestType, Message
onCompleteMsg)
private DataConnection checkForCompatibleConnectedApnContext(ApnContext apnContext)
```

8.3 APIs for Apps

The following clauses shows examples how Apps can invoke APIs to use a DNN to select a network slice according to URSP rules.

8.3.1 Solution 1

8.3.1.1 Request network with NET_CAPABILITY_DNN1

```
private void requestNetwork() {
  if (mConnectivityManager == null) {
    mConnectivityManager = (ConnectivityManager) mContext.getSystemService(
        Context.CONNECTIVITY_SERVICE);
  }
  mNetworkCallback = new NetworkRequestCallback();
  Builder builder = new NetworkRequest.Builder();
  builder.addTransportType(NetworkCapabilities.TRANSPORT_CELLULAR);

  try {
    final Field ncDnn = NetworkCapabilities.class.getDeclaredField("NET_CAPABILITY_DNN1");
    builder.addCapability(((Integer))ncDnn.get(mConnectivityManager));
  } catch (ClassNotFoundException e) {
    Log.e(TAG, e.toString());
  } catch (NoSuchFieldException e) {
        // fallback to Internet if failed to connect to network with NET_CAPABILITY_DNN1
    Log.e(TAG, e.toString());
    builder.addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET);
  }

  NetworkRequest nwRequest = builder.build();
  mConnectivityManager.requestNetwork(nwRequest, mNetworkCallback, TIME_OUT);
}
```

8.3.1.2 Realize network callback method for binding a socket

```
class NetworkRequestCallback extends ConnectivityManager.NetworkCallback {
  public NetworkRequestCallback() {
  }
  @Override
```

```
    public void onAvailable(Network network) {
      Log.d(TAG, "onAvailable");
      synchronized (mConnectLock) {
        mNetwork = network;
        mConnectLock.notifyAll();
        try {
          mSocket = mNetwork.getSocketFactory().createSocket();
          mSocket.connect(new InetSocketAddress("xxxx", xxxx));
        } catch(Exception e) {
          Log.d(TAG, "socket connect fail! e = " + e);
        }
      }
    }
    @Override
    public void onLost(Network network) {
      Log.d(TAG, "onLost");
      releaseNetwork();
    }
  }
```

8.3.2 Solution 2：

8.3.2.1 Request network with NET_CAPABILITY_DNN and include DNN string in networkSpecifier

```
    private void requestNetwork() {
      if (mConnectivityManager == null) {
        mConnectivityManager = (ConnectivityManager) mContext.getSystemService(
            Context.CONNECTIVITY_SERVICE);
      }
      mNetworkCallback = new NetworkRequestCallback();
      Builder builder = new NetworkRequest.Builder();
      builder.addTransportType(NetworkCapabilities.TRANSPORT_CELLULAR);

      try {
        final Field ncDnn = NetworkCapabilities.class.getDeclaredField("NET_CAPABILITY_DNN");
        builder.addCapability(((Integer))ncDnn.get(mConnectivityManager));
```

```
            String className = "android.net.DnnNetworkSpecifier";
            Class<?> dnnNS = Class.forName(className);
            builder.setNetworkSpecifier(dnnNS.getDeclaredConstructor().newInstance(mSubId, "DNN1"));
        } catch (Exception e) {
                    // fallback to Internet if failed to connect to network with NET_CAPABILITY_DNN
            Log.e(TAG, e.toString());
            builder.addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET);
        }
        NetworkRequest nwRequest = builder.build();
        mConnectivityManager.requestNetwork(nwRequest, mNetworkCallback, TIME_OUT);
    }
```

8.3.2.2 Realize network callback method for binding a socket

```
class NetworkRequestCallback extends ConnectivityManager.NetworkCallback {
    public NetworkRequestCallback() {
    }
    @Override
    public void onAvailable(Network network) {
        Log.d(TAG, "onAvailable");
        synchronized (mConnectLock) {
            mNetwork = network;
            mConnectLock.notifyAll();
            try {
                mSocket = mNetwork.getSocketFactory().createSocket();
                mSocket.connect(new InetSocketAddress("xxxx", xxxx));
            } catch(Exception e) {
                Log.d(TAG, "socket connect fail! e = " + e);
            }
        }
    }
    @Override
    public void onLost(Network network) {
        Log.d(TAG, "onLost");
        releaseNetwork();
    }
}
```