Deep-Dive into SCP Model-D Based Communication

Contents

Introduction

This document describes the deep dive of SCP Model-D Communication approach between Cisco AMF/SMF and third party NF.

Prerequisites

Requirements

Cisco recommends that you have knowledge of these topics:

- Functionality of Access and Mobility Management Function (AMF)
- Functionality of Session Management Function (SMF)
- Functionality of Service Communication Proxy (SCP)

Components Used

This document is not restricted to specific software and hardware versions.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Background Information

Operators around the world can choose between multiple communication models using SCP for the Network Function (NF) discovery and subsequent NF to NF communications. This topic touches upon concepts around various communication models and the call flow/configuration changes required at Subscriber Microservices Infrastructure (SMI), AMF/SMF in order to have SCP Model-D based communication.

Architecture and Solution Overview

In the Service based architecture (SBA), the SCP acts as an intermediary, facilitating indirect communication between NFs by handling routing, load balancing, and service discovery, ultimately streamlining the service-based architecture.

3GPP 23.501 Annex-E details the four communication models between NF in a 5GC deployment.

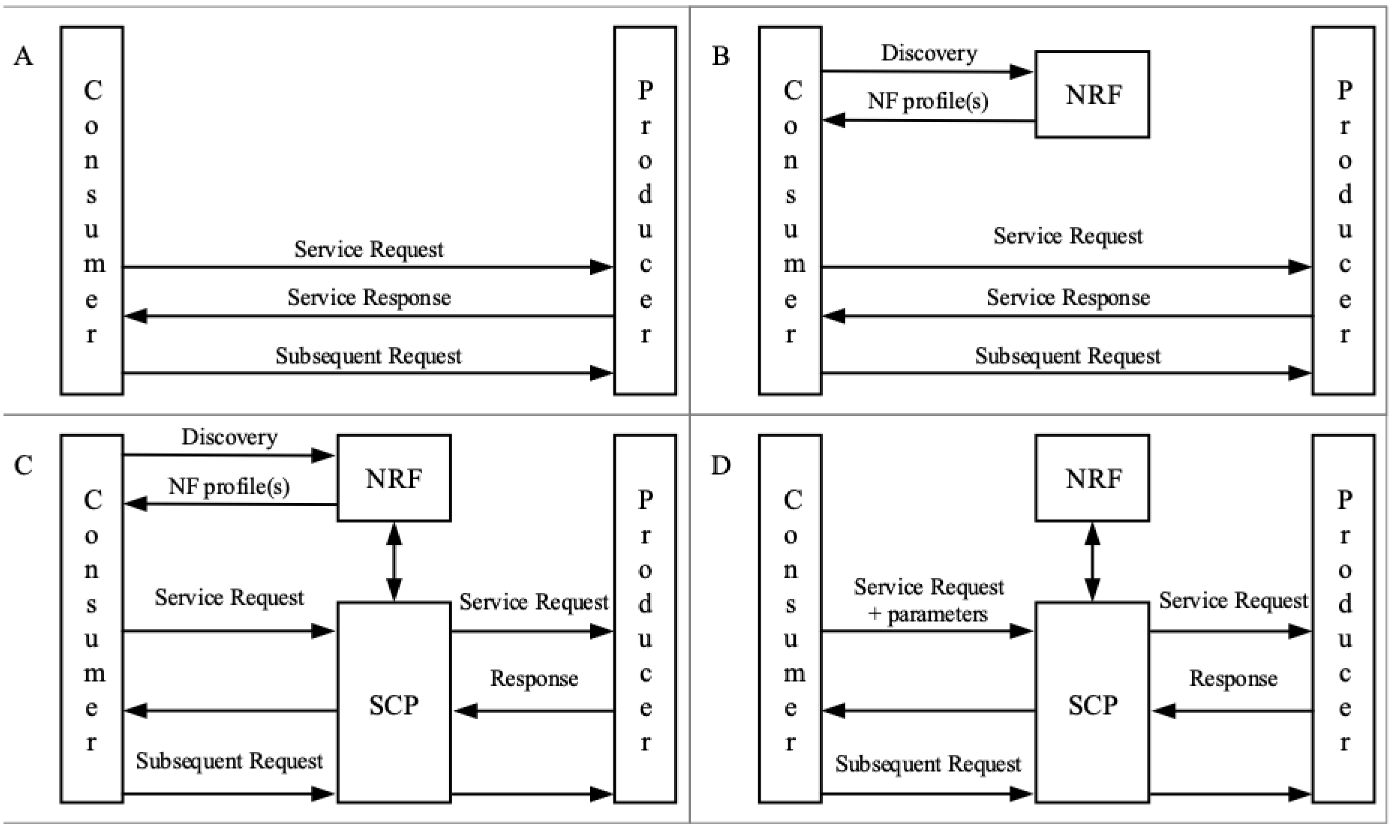


Figure A: (Different communication Models involving SCP)

Model-A - Direct communication without Network Repository Function (NRF) interaction: Consumers are configured with the 'NF profiles' of the producers and directly communicate with a producer of their choice. This is type of static selection and neither NRF nor SCP are used.

Model-B - Direct communication with NRF interaction: Consumers do discovery by querying the NRF. Based on the discovery result, the consumer does the selection. The consumer sends the request to the selected producer.

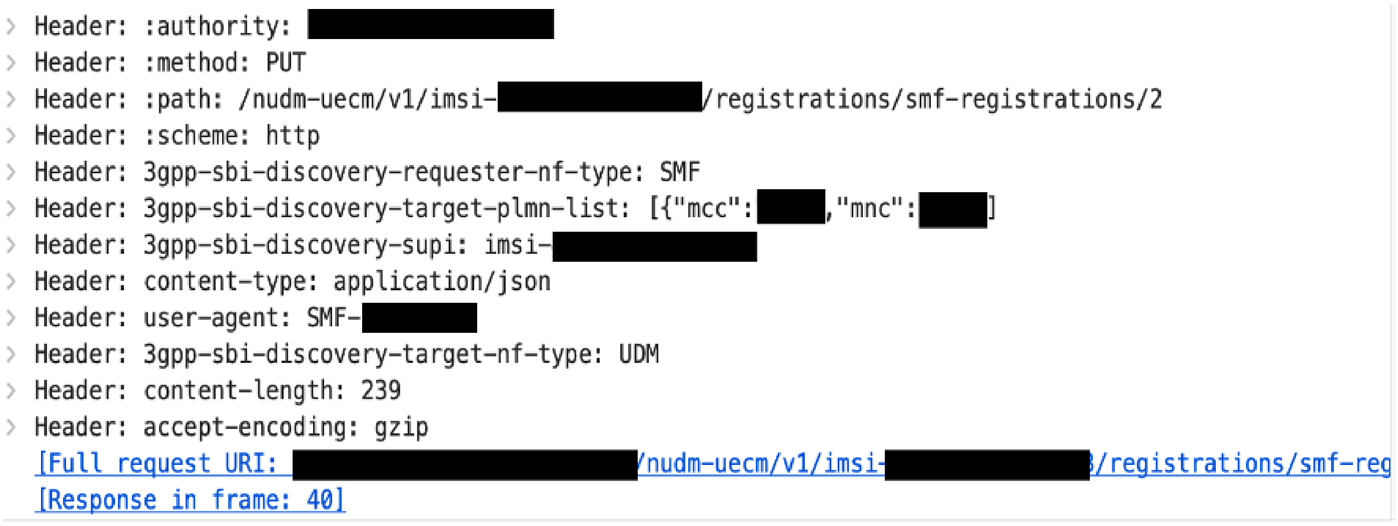
Model-C - Indirect communication without delegated discovery: Consumers discover by querying the NRF. Based on discovery result, the consumer does the selection of an NF set or a specific NF instance of NF set. The consumer sends the request to the SCP containing the address of the chosen service producer pointing to a NF service instance or a set of NF service instances. In the latter case, the SCP chooses an NF Service instance. If possible, the SCP interacts with NRF in order to get selection parameters such as location, capacity, and so on. The SCP routes the request to the chosen NF service producer instance.

Model-D - Indirect communication with delegated discovery: Consumers are not involved in discovery or selection. The consumer adds any necessary discovery and selection parameters required to find a suitable producer to the service request. The SCP uses the request address and the discovery and selection parameters in the request message in order to route the request to a suitable producer instance. The SCP can perform discovery with an NRF and obtain a discovery result.

Deep dive on Model-D based Communication: When Call Model-D is used, the NF consumer does not directly send a request to the NRF but delegate to the SCP this discovery. The NF client sends a message to the SCP and concatenate for each of these discovery factors the string '3gpp-sbi-discovery' with the name of the Discovery factor which will be used if NF discovery will be done via the NRF.

For a scenario, where SMF will be looking for Unified Data Management (UDM) with service-names nudm-sdm, the discovery factors will be passed to the SCP:

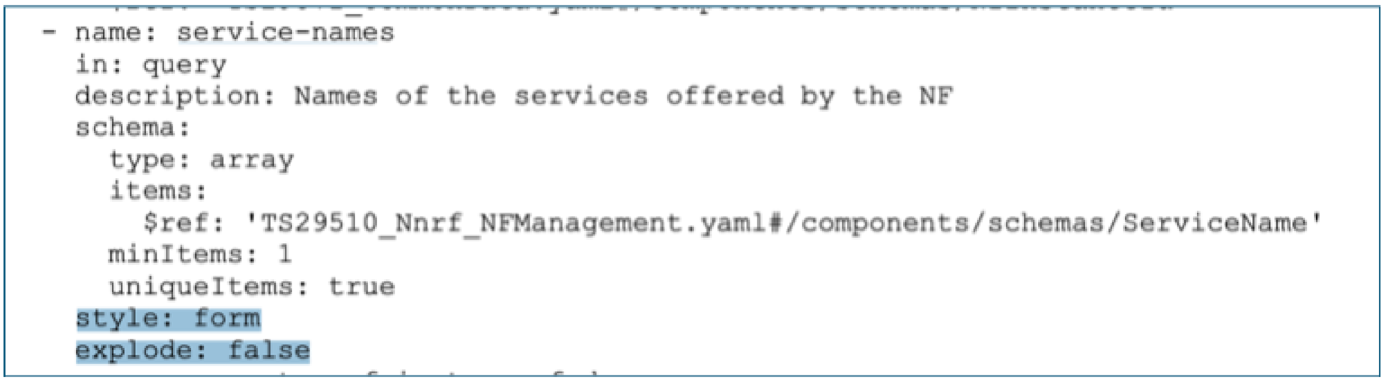
- Authority Header: the authority carries either the Fully Qualified Domain Name (FQDN) or the IP address, with priority given to the IP address configuration.
- 3gpp-sbi-discovery-requester-nf-type: SMF
- 3gpp-sbi-discovery-target-nf-type: UDM
- 3gpp-Sbi-discovery-service-name: nudm-sdm



```
> Header: :authority: [redacted]
> Header: :method: PUT
> Header: :path: /nudm-uecm/v1/imsi-[redacted]/registrations/smf-registrations/2
> Header: :scheme: http
> Header: 3gpp-sbi-discovery-requester-nf-type: SMF
> Header: 3gpp-sbi-discovery-target-plmn-list: [{"mcc":[redacted],"mnc":[redacted]]
> Header: 3gpp-sbi-discovery-supi: imsi-[redacted]
> Header: content-type: application/json
> Header: user-agent: SMF-[redacted]
> Header: 3gpp-sbi-discovery-target-nf-type: UDM
> Header: content-length: 239
> Header: accept-encoding: gzip
  [Full request URI: [redacted]/nudm-uecm/v1/imsi-[redacted]/registrations/smf-reg
  [Response in frame: 40]
```

Figure B: (SMF-UDM Communication via SCP model D)

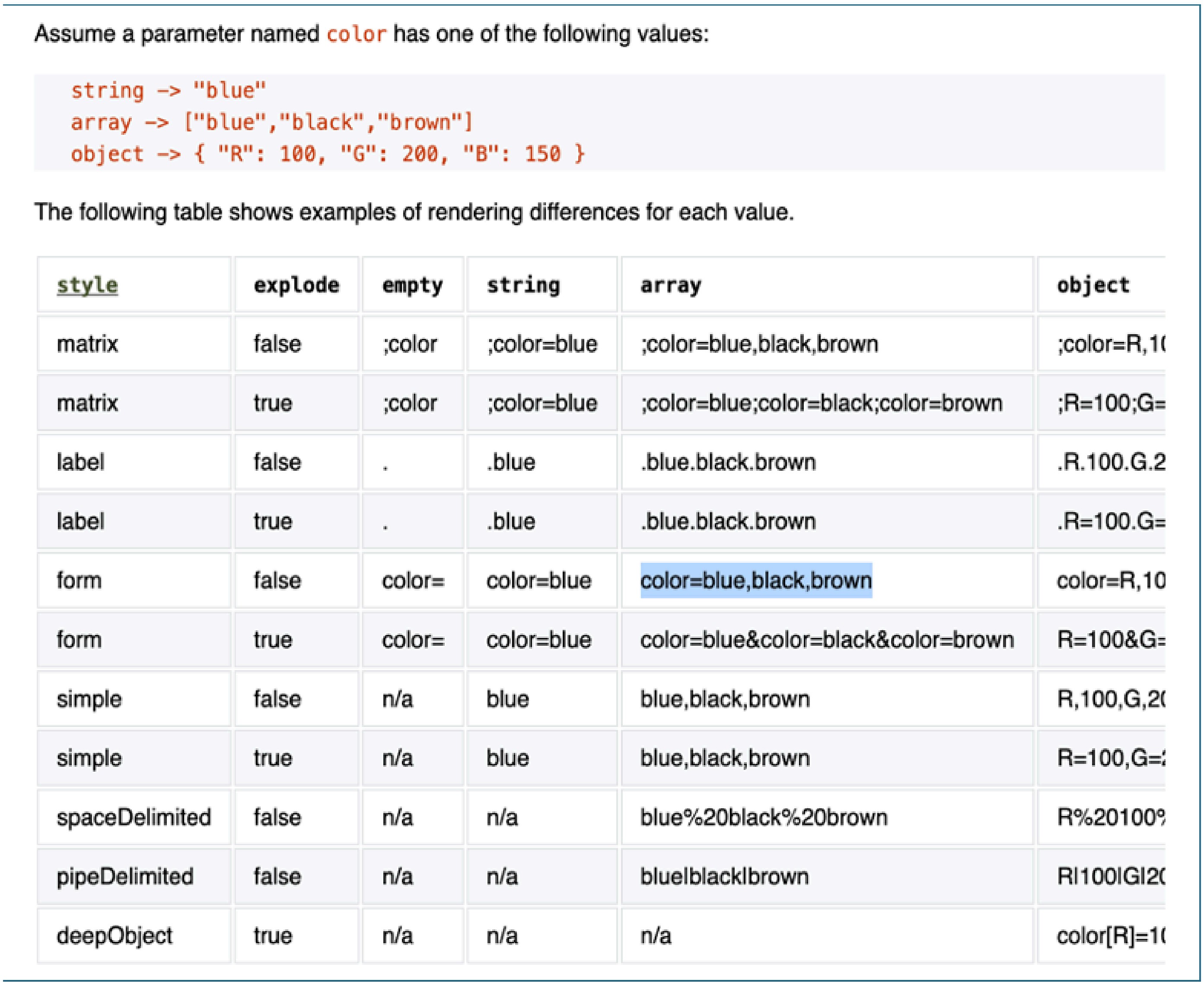
Note: The 3gpp-sbi-discovery-service name format is in plain string format and not in array format as per 3gpp 29.510 and open API definitions (4.7.12.4 Style). In the 29.510 3gpp-sbi-discovery-service-name is mentioned as an array format.



```
- name: service-names
  in: query
  description: Names of the services offered by the NF
  schema:
    type: array
    items:
      $ref: 'TS29510_Nnrf_NFManagement.yaml#/components/schemas/ServiceName'
    minItems: 1
    uniqueItems: true
  style: form
  explode: false
```

Figure C: (Snapshot from 29.510 Spec)

However the style:form and explode:false converts the array into a plain string which is explained by taking an example from OpenAPI.



Assume a parameter named `color` has one of the following values:

```
string -> "blue"
array -> ["blue","black","brown"]
object -> { "R": 100, "G": 200, "B": 150 }
```

The following table shows examples of rendering differences for each value.

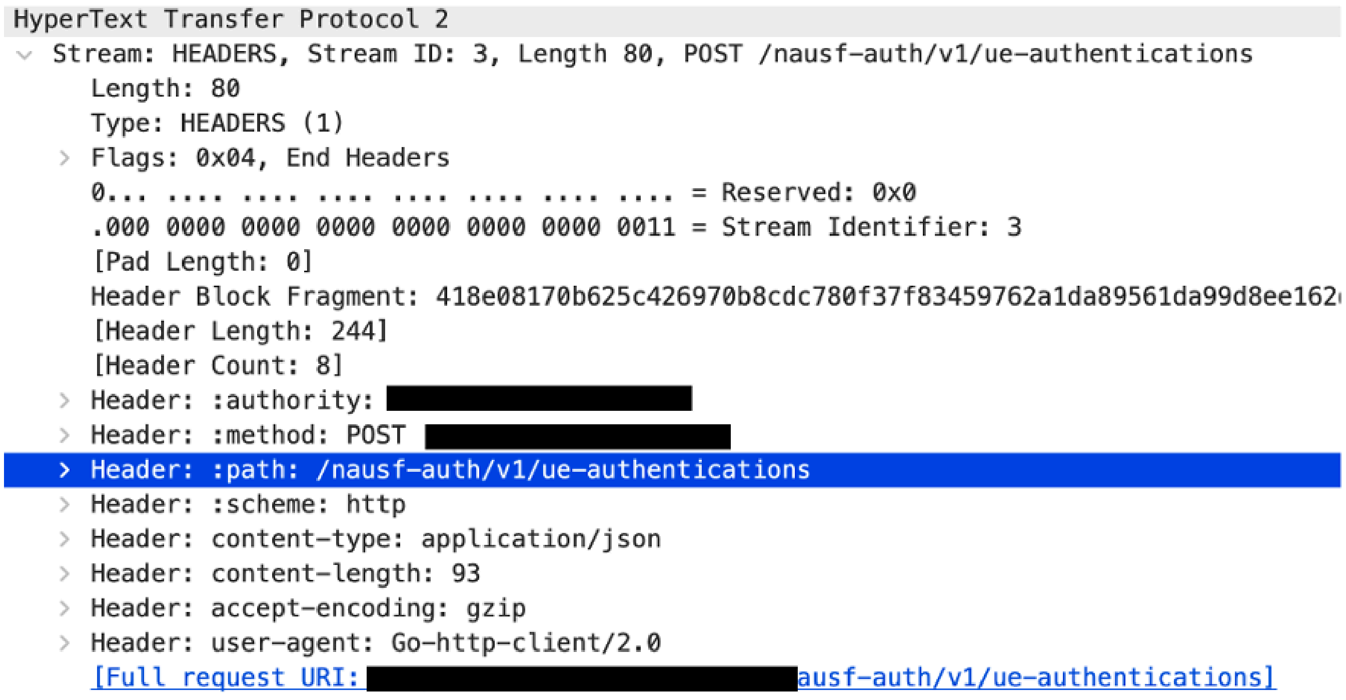
style	explode	empty	string	array	object
matrix	false	;color	;color=blue	;color=blue,black,brown	;color=R,1
matrix	true	;color	;color=blue	;color=blue;color=black;color=brown	;R=100;G=
label	false	.	.blue	.blue.black.brown	.R.100.G.2
label	true	.	.blue	.blue.black.brown	.R=100.G=
form	false	color=	color=blue	color=blue,black,brown	color=R,10
form	true	color=	color=blue	color=blue&color=black&color=brown	R=100&G=
simple	false	n/a	blue	blue,black,brown	R,100,G,2
simple	true	n/a	blue	blue,black,brown	R=100,G=
spaceDelimited	false	n/a	n/a	blue%20black%20brown	R%20100
pipeDelimited	false	n/a	n/a	blue|black|brown	R|100|G|2
deepObject	true	n/a	n/a	n/a	color[R]=1

Figure D: (Snapshot from Open API: (4.7.12.4 Style Examples))

You have CLI control in both AMF and SMF to send the parameter 3gpp-sbi-discovery-service as this is optional (can be done depending upon deployment environment).

In case of Model-B, if you take the example of AMF and Authentication Server Function (AUSF) communication, then once AUSF is discovered, the AMF sends the POST to AUSF with AUSF IP/FQDN and Port.

POST http://<ausf-fqdn>:<port>/nausf-auth/v1/ue-authentications.



```
HyperText Transfer Protocol 2
∨ Stream: HEADERS, Stream ID: 3, Length 80, POST /nausf-auth/v1/ue-authentications
    Length: 80
    Type: HEADERS (1)
  > Flags: 0x04, End Headers
    0... .... .... .... .... .... .... .... = Reserved: 0x0
    .000 0000 0000 0000 0000 0000 0000 0011 = Stream Identifier: 3
    [Pad Length: 0]
    Header Block Fragment: 418e08170b625c426970b8cdc780f37f83459762a1da89561da99d8ee162
    [Header Length: 244]
    [Header Count: 8]
  > Header: :authority: [redacted]
  > Header: :method: POST [redacted]
  > Header: :path: /nausf-auth/v1/ue-authentications
  > Header: :scheme: http
  > Header: content-type: application/json
  > Header: content-length: 93
  > Header: accept-encoding: gzip
  > Header: user-agent: Go-http-client/2.0
    [Full request URI: [redacted]ausf-auth/v1/ue-authentications]
```

Figure E: (AMF-AUSF Communication via Model B)

In Model-D, as the discovery is performed by the SCP, instead of POST http(s)://<ausf-fqdn>:<ausf-port>/nausf-auth/v1/ue-authentications the AMF sends the modified POST request which is:

POST http(s)://<scp-fqdn>:<scp-port>/nausf-auth/v1/ue-authentications

Or

POST http(s)://<scp-fqdn>:<scp-port>/nscp-route/nausf-auth/v1/ue-authentications(if apiroot=nscp-route)

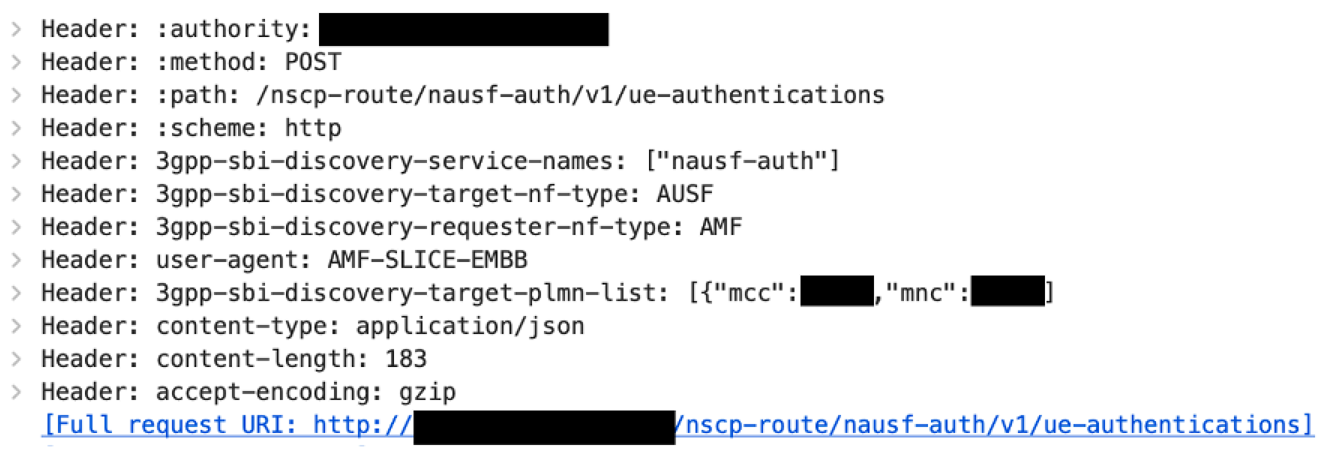
With

3gpp-Sbi-Discovery-target-nf-type: AUSF

3gpp-Sbi-Discovery-Preferred-locality: LOC1

3gpp-Sbi-Discovery-service-name

Where you can see that AMF has replaced the api-root (<ausf-fqdn>:<ausf-port>) of the AUSF with the api-root of the SCP.



```
> Header: :authority: [redacted]
> Header: :method: POST
> Header: :path: /nscp-route/nausf-auth/v1/ue-authentications
> Header: :scheme: http
> Header: 3gpp-sbi-discovery-service-names: ["nausf-auth"]
> Header: 3gpp-sbi-discovery-target-nf-type: AUSF
> Header: 3gpp-sbi-discovery-requester-nf-type: AMF
> Header: user-agent: AMF-SLICE-EMBB
> Header: 3gpp-sbi-discovery-target-plmn-list: [{"mcc":[redacted],"mnc":[redacted]]
> Header: content-type: application/json
> Header: content-length: 183
> Header: accept-encoding: gzip
  [Full request URI: http://[redacted]/nscp-route/nausf-auth/v1/ue-authentications]
```

Figure F: (AMF-AUSF Communication via SCP-Model D)

The 3gpp-sbi-discovery parameters allows the SCP to retrieve the best NF and then forward the POST request where it replaces the api-root of the SCP with the api-root received from the NRF after having received the response to its discovery request.

Configurations Required at AMF/SMF

In order to indicate for each NF (for example, UDM) which call-model must be used, the nf-selection-model configuration is used within the associated 'profile network-element'.

<#root>

```
profile network-element udm prf-udm-scp


 [...]




nf-selection-model priority <>[local | nrf-query | nrf-query-peer-input | nrf-query-and-scp | scp]


exit
```

Once Model-D is chosen, the query-params configured for the associated network-element are still used and are passed to the SCP in the format '3gpp-Sbi-Discovery-<query-param>'.

<#root>

```
[smf] smf(config)# profile network-element udm prf-udm-scp
```

```
[smf] smf(config-udm-udm1)# query-params

Possible completions:

 [  chf-supported-plmn  dnn  requester-snssais  tai  target-nf-instance-id  target-plmn ]
```

Eventually the profile network-element is mapped to the profile Data Network Name (dnn).

<#root>

```
profile dnn ims

 network-element-profiles udm prf-udm-scp

 network-element-profiles scp prf-scp

exit
```

SCP(s) are defined as network-element.

nf-client-profile and a failure-handling profile is mapped with network-element.

<#root>

```
profile network-element scp <>

 nf-client-profile        <>

 failure-handling-profile <>
```

```
exit
```

The nf-client-profile of type scp-profile details the characteristics of the SCP endpoint.

Here nscp-route can be added in api-root.

<#root>

```
profile nf-client nf-type scp

 scp-profile <>

  locality LOC1

   priority 30

   service name type <>

    responsetimeout 4000

    endpoint-profile EP1

     capacity   30

     api-root   nscp-route

     priority   10

     uri-scheme http
     endpoint-name scp-customer.com

      priority 10

      capacity 50

      primary ip-address ipv4

      primary ip-address port

      fqdn name <>

      fqdn port <>


exit
```

SMF FQDN is configured in endpoint southbound interface (SBI).

<#root>

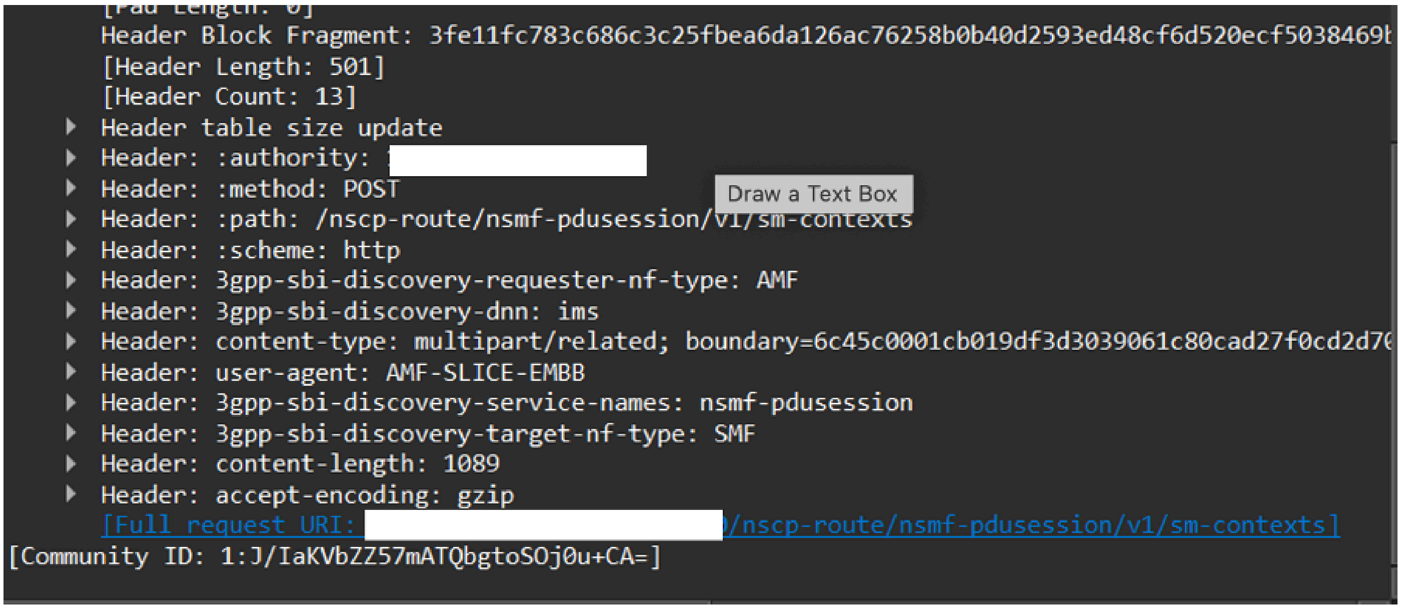
```
endpoint sbi

relicas 2

nodes 2

fqdn <>
```

Sample Snap of Packets



```
        [Pad Length: 0]
        Header Block Fragment: 3fe11fc783c686c3c25fbea6da126ac76258b0b40d2593ed48cf6d520ecf5038469
        [Header Length: 501]
        [Header Count: 13]
     ▶  Header table size update
     ▶  Header: :authority: [redacted]
     ▶  Header: :method: POST
     ▶  Header: :path: /nscp-route/nsmf-pdusession/v1/sm-contexts
     ▶  Header: :scheme: http
     ▶  Header: 3gpp-sbi-discovery-requester-nf-type: AMF
     ▶  Header: 3gpp-sbi-discovery-dnn: ims
     ▶  Header: content-type: multipart/related; boundary=6c45c0001cb019df3d3039061c80cad27f0cd2d7
     ▶  Header: user-agent: AMF-SLICE-EMBB
     ▶  Header: 3gpp-sbi-discovery-service-names: nsmf-pdusession
     ▶  Header: 3gpp-sbi-discovery-target-nf-type: SMF
     ▶  Header: content-length: 1089
     ▶  Header: accept-encoding: gzip
        [Full request URI: [redacted]/nscp-route/nsmf-pdusession/v1/sm-contexts]
[Community ID: 1:J/IaKVbZZ57mATQbgtoSOj0u+CA=]
```

Figure G: (AMF- SMF nsmf-pdusession communication via SCP Model D)

You need from the profile dnn to refer to the SCP network element just configured.

<#root>

```
profile dnn <>



 network-element-profiles udm <>



 network-element-profiles scp <>



exit
```

If SCP failure-handling is configured with action as retry, SMF attempts alternate SCP based on SCP configuration and retry count.

If SCP failure-handling is configured with action as retry-and-fallback for a particular service name and message type then the fallback to Model-A happens.

This ailure Handling Profile for SCP (FHSCP) is used if the error is trigerred from the SCP (server header indicating SCP) and the NF-client configuration for the peer is present.

<#root>

```
profile nf-client-failure nf-type scp



   profile failure-handling <>



   service name type npcf-smpolicycontrol



    responsetimeout 1800



    message type PcfSmpolicycontrolCreate



    status-code httpv2 0,307,429,500,503-504



     retry  1



     action retry-and-fallback



     exit
```

Example of the nf-client profile for Policy Control Function (PCF) for the scenario where action retry and falback is configured for the message type PcfSmpolicycontrolCreate:

<#root>

```
profile nf-client nf-type pcf



 pcf-profile <>



  locality LOC1
```

```
priority 1

service name type npcf-smpolicycontrol

 endpoint-profile epprof

  capacity   10

  priority   1

  uri-scheme http

  endpoint-name ep1

   priority 1

   capacity 10

   primary ip-address ipv4 <>

   primary ip-address port <>

  exit

  endpoint-name ep2

   priority 1
}

   capacity 10

   primary ip-address ipv4 <>

   primary ip-address port <>

  exit
```

Core DNS PODs and Configuration Required at SMI Layer

CoreDNS pods, which are part of the kube-system namespace, are deployed as a 2-pod replicaset. These pods can be scheduled on any of the two master/control nodes and are not dependent on where the nameserver IP is configured in cluster manager.

However, it is recommended to configure the nameserver IP in all of the control/master nodes as you do not have a labelling control to spin the CoreDNS pods as per your wish. If the route to nameservers are not present on any of the master where CoreDNS is deployed then the SMF/AMF cluster sync gets failed.

Currently, CoreDNS forwards DNS requests to the nameserver specified in the nodes **resolv.conf** file.

'kubectl edit configmap coredns -n kube-system' you have:

<#root>

```
 {

forward ./etc/resolv.conf{

     max_concurrent 1000

  }
```

When checking **/etc/resolv.conf** on the master node where service is started, it must contain:

<#root>

```
 name server <>

     name server <>
```

Sample nameserver configuration in master/control node:

<#root>

```
nodes <>

initial-boot netplan vlans <>

dhcp4 false

dhcp6 false

addresses [<>]

nameserver addresses [<>]

id <>

link <>

exit
```