RADIUS-Triggered Dynamic Data Service Provisioning

In This Chapter

This section describes advanced RADIUS-triggered dynamic data service provisioning configurations.

Topics in this section include:

- Applicability on page 2606
- Overview on page 2607
- Configuration on page 2610
- Conclusion on page 2649

Applicability

This example is applicable to all 7750 SR-7/12 and 7450 ESS-7/12 in mixed mode with multicore CPM (CPM-3 and later) and ESM capability.

This feature is not supported on the 7950 XRS, 7750 SR-1, 7450 in pure ESS-mode or 7710 SR.

The configuration was tested on release 11.0R2.

Overview

RADIUS-triggered dynamic data services enables a zero touch, single-ended provisioning model for business services on the basis of Enhanced Subscriber Management functionality.

Triggered by the authentication of a single or dual stack PPPoE session or single stack IPv4 host as the “control channel” from the business CPE, parameters are passed in a RADIUS Access Accept or Change of Authorization (CoA) message to set up one or multiple Layer 2 or Layer 3 data services.

This concept removes the need to have an Operations Support System (OSS) responsible for the service provisioning and is particularly beneficial in a highly dynamic network environment, where physical network topologies – especially in the access – change frequently. With a regular service provisioning, frequent changes would be hard to keep track of. In the RADIUS-based model the service gets instantiated wherever it “pops-up” in the network. Even planned customer moves to a different office would not require advanced notifications and lead times but could be instantaneous, assuming the pure physical connectivity is given.

A variation of the current service offering will only require one or a few modified service parameters in the RADIUS user database and does not require timely and costly IT changes (for RADIUS those service parameters are just attributes; the RADIUS server does not check the logic). This speeds up the time-to-market for new service offerings, which is another big advantage.

Taking this logic to its full extent, it becomes immediately clear that the managed business CPE terminating the carrier service (and being responsible for the PPP or DHCP control channel) also needs to be provisioned in the most flexible way. Through the control channel or a dedicated management channel instantiated as the first dynamic data service, the business CPE should get its full configuration from a configuration server via a pre-populated configuration file. The details of the CPE provisioning are outside of the scope of this example and therefore not discussed further.

As the whole approach is centered around the principle of “highly flexible in a highly dynamic environment”, it is naturally required to maintain as little state information about connections in the RADIUS parameter attributes as possible. For example, fixed remote peer IP-addresses for the SDPs used in a VPLS service in the RADIUS parameter lists would remove all the flexibility and would not allow access services to be moved dynamically. As such the allowed data services for this functionality focuses on those types where a control protocol like BGP is used to exchange VPN membership information. Dynamic data services supported in this release include local Epipe VLL services, Epipe VLL services with dynamic Multi-Segment PseudoWires (MS-PWs) (FEC129), VPLS services with BGP-AD PWs, IES, and VPRN services.

A Python script interface adds a flexible abstraction layer so that only the business user specific service parameters (service type, IP address, QoS and filter parameters, etc.) are required from RADIUS and are then used in a CLI template to set up the target service.

The setup sequence is shown in Figure 398 with the example of a VPLS service.

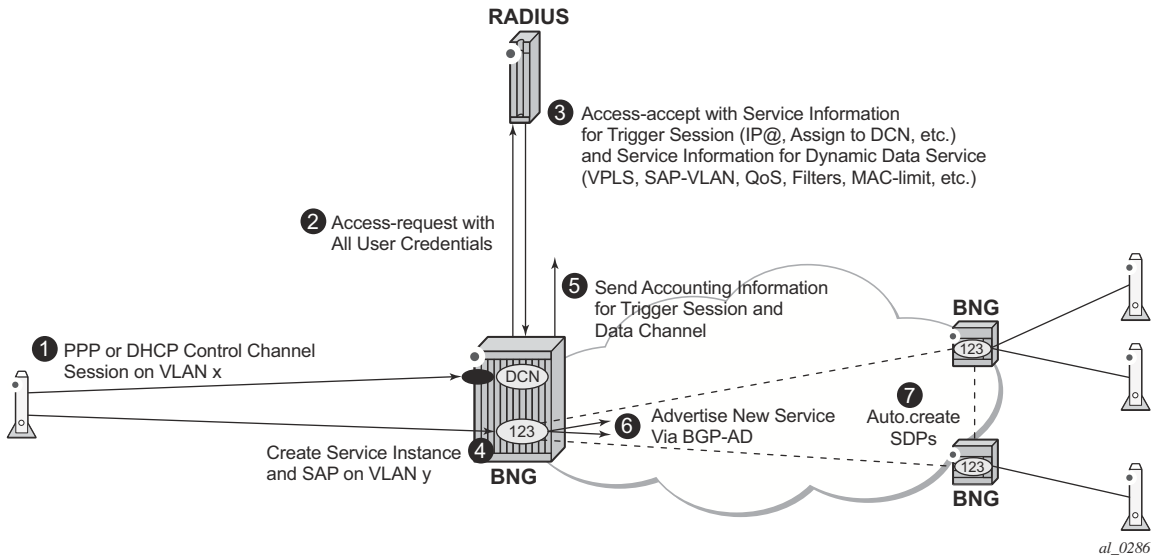

Figure 398: Principle Model of Dynamic Data Services

1. Business CPE initiates a PPP or DHCP “control channel” session. This session is important to let the BNG and RADIUS understand the existence of a new access circuit and the location of the service endpoint.
2. BNG sends an Access-Request with all user credentials to RADIUS.
3. RADIUS replies using an Access-Accept with attributes for the PPP/DHCP control channel and attributes for the dynamic data service (Service-type, SAP-VLAN, QoS, Filter, etc.).
4. BNG creates a dynamic data service instance (if it is first access-circuit for this service) and a SAP, thus completing the service configuration.
5. BNG sends an Accounting Start for PPP/DHCP control channel session and also for the dynamic data service session (and subsequently interim accountings and accounting stop for both).
6. BNG advertises VPLS instance-ID (123) via BGP-AD to other PEs/BNGs.
7. PEs/BNGs with same service instance will auto-establish SDPs to the BNG.

The result is a fully functional service which is the same as a traditionally configured service.

The lifetime of the dynamic data services are bound to the existence of the control channel session. If, for whatever reason, the control channel session is torn down all associated dynamic data services will also be terminated.

Dynamic data service SAPs have to be located on dot1q or qinq encapsulated Ethernet ports and can be part of a LAG.

Both XML accounting and RADIUS accounting can be enabled on a dynamic data service SAP. The RADIUS accounting data can be sent to up to two different RADIUS servers.

There is a strict separation of services created by dynamic service provisioning and services created via the CLI or through other standard mechanisms (5620 SAM, SNMP). It is therefore not allowed to:

- create a dynamic services object in a local provisioned CLI/SNMP context (e.g. create a dynamic SAP in a local provisioned VPLS).
- create a local provisioned object in a dynamic service context (e.g. create a SAP via CLI/SNMP in a dynamic VPLS service).
- change parameters in a local provisioned CLI/SNMP context using the dynamic services model (change system name with dynamic services provisioning).
- change parameters with the CLI/SNMP in a dynamically created context.
- delete a local provisioned object using dynamic provisioning model.
- delete a dynamic provisioning object using the CLI/SNMP.
- create a reference to a dynamic services object in a local provisioned CLI/SNMP context (reference to dynamic interface in **router ospf**)

A special command exists to overcome some of the above rules. This command is designed to ease Python script creation and testing and not for normal operations. This is discussed in Configuration on page 2610.

Configuration

It is assumed that the reader is familiar with the regular Enhanced Subscriber Management (ESM) functionality as well as with general service related configurations. Furthermore certain knowledge about Python programming is also assumed.

The test topology is shown in Figure 399.

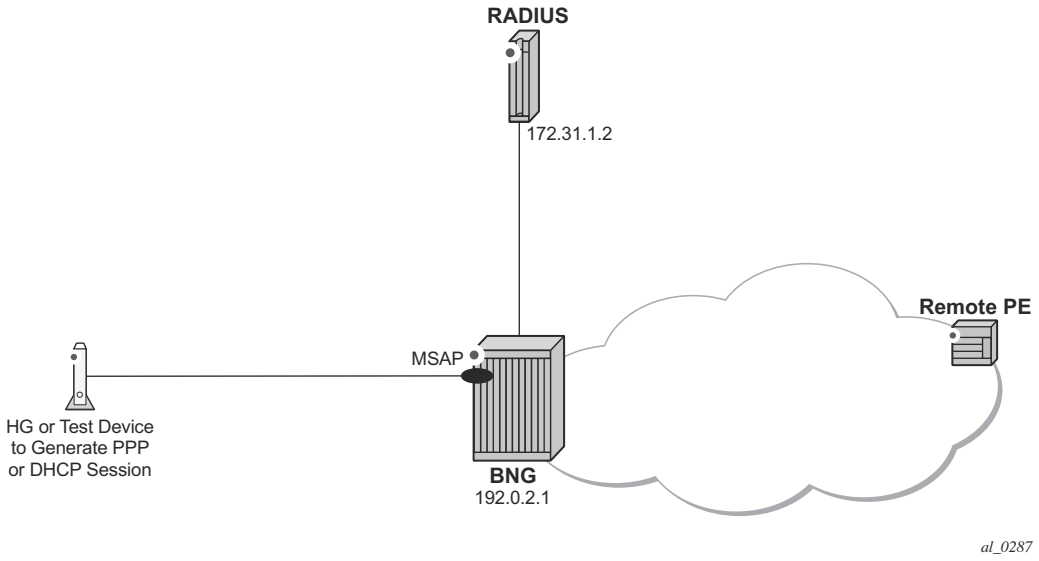

Figure 399: Test Topology

The pure service setup can be tested with a single node acting as BNG. However an Epipe service between two nodes will not normally become status “up” with only one endpoint in an up state. As such, for packets should be sent through the established dynamic data service, a remote PE could also be configured. The remote PE could have its data service configured in a regular fashion, meaning via CLI/SNMP or 5620 SAM.

The required functionality on the BNG is divided into multiple building blocks. The following sections discuss each building block in detail.

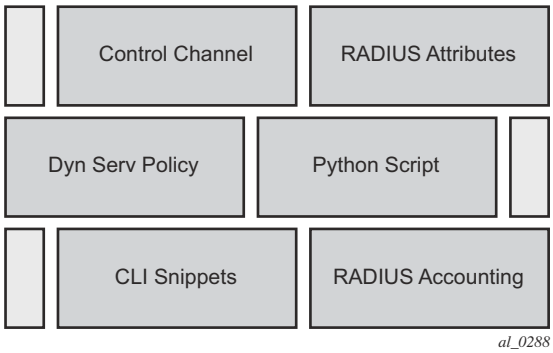

Figure 400: Building Blocks of Dynamic Data Services

Based on a PPP or DHCP control session, RADIUS will return the required parameters for the dynamic data service via dedicated Vendor Specific Attributes (VSAs). The existence of those attributes in the RADIUS Accept message will trigger the relaying of the parameters relating to those attributes towards the Python script defined in the dynamic service policy, which will process them to generate the regular CLI output for the various service types (IES, VPRN, Epipe, VPLS).

For efficiency and flexibility the Python script needs to be structured into different parts per service which then reference each other internally. Those parts are called snippets.

Finally, as the services are initiated from RADIUS, RADIUS accounting messages per dynamic data service will be sent to the RADIUS server as a necessary feedback mechanism to inform the RADIUS server about a successful or failed service setup.

Building Block: Control Channel

The configuration to authenticate and instantiate a dynamic data service control channel is identical to a residential Enhanced Subscriber Management (ESM) configuration. Examples for this can be found in other sections of the advanced configuration guide and will not be covered here in detail.

Building Block: Dynamic Services Policy

The dynamic services parameters are configured under the **configure service dynamic-services** CLI context. The following output shows two policy examples.

```
configure service dynamic-services
             dynamic-services-policy "dynamic-services-1" create
               accounting-1
                   server-policy "radius-server-policy-1"
                   update-interval min 5
               exit
               accounting-2
                   server-policy "radius-server-policy-2"
```

```
            stats-type time
            update-interval min 5
            update-interval-jitter absolute 10
        exit
        cli-user "dynuser"
        description "Dynamic Service Policy #1"
        sap-limit 4000
        script-policy "script-policy-1"
    exit
    dynamic-services-policy "dynamic-services-2" create
        accounting-1
            server-policy "radius-server-policy-2"
      stats-type volume-time
            update-interval min 30
            update-interval-jitter absolute 20
        exit
        accounting-2
            server-policy "radius-server-policy-2"
            stats-type time
            update-interval min 5
            update-interval-jitter absolute 10
        exit
        cli-user "dynuser"
        description "Dynamic Service Policy #2"
        sap-limit 100
        script-policy "script-policy-2"
    exit
    service-range 1000 10000
    timers
        setup-timeout access-accept 3
    exit
```

Details of each command and the possible parameters can be found in the SR OS Triple Play Guide in the RADIUS Triggered Dynamic Data Services section.

On the top command level under the dynamic-services sub-tree there are three options:

- dynamic-services-policy
- service-range
- timers

The setup-timeout value under **timers** is used to limit the maximum delay allowed for a dynamic data service setup. In addition, it also protects the node during times where there is a high load on the CPU. If a requested dynamic data service cannot be established in the specified time the request will be dropped.

Dynamic data services are not preferred over regular ESM subscribers. As such, given a BNG with a mix of residential ESM subscribers and business customers with dynamic data services, all compete for the same CPU resources to establish the connections.

However, dynamic data services are expected to have a very long lifetime compared to potentially very dynamic lifetimes for residential subscribers. In a regular operating mode the amount of

additional setup requests for dynamic data services should be relatively small. Only in the event of a node reboot will all users again compete to gain access, where longer setup-times are inevitable.

The service-range value reserves a certain amount of service IDs for the use of dynamic data services. The configured range is no longer available for regular provisioned services configured via the CLI/SNMP.

The dynamic-services-policy contains a CLI-user identifier, SAP-limits, accounting parameters and reference to a Python script policy which is used when creating a dynamic data service. Multiple dynamic services policies can be created to enable different profiles to be used for different users/customers or services (as an example, two different departments within the service provider, one responsible for Layer 2 services, one for Layer 3 services). The policy used for a dynamic data service is determined from the Alc-Dyn-Serv-Policy [26-6527-167] RADIUS attribute. If the attribute is not present and a policy named **default** exists, then the **default** policy is used, otherwise the dynamic data service creation fails.

Up to two accounting server policies can be defined. This allows the use of separate RADIUS accounting servers independent from the accounting servers used for residential services. The parameters defined in the accounting sections are the default values which are used if no specific values are sent via RADIUS VSAs.

As the service is established via RADIUS, a feedback mechanism towards RADIUS is most likely required which would be at least RADIUS start and stop messages per service/session. In addition performance counters (with a fixed set of parameters) can also be included in the RADIUS messages. It is also possible to use the standard service-accounting under the service instance and remove any counters from the RADIUS accounting messages.

The specification of a CLI user allows linking of the dynamic data service to a specific user-profile. In addition, this facilitates limiting of the scope of allowed service configurations even further, based on the specified context under the user profile.

The CLI-user needs to be configured locally on the node and needs to have a local user profile (remote authorization via TACACS/RADIUS is not possible).

The radius-script-policy is configured under the **configure aaa** CLI context.

```
configure aaa
radius-script-policy "script-policy-2"
     action-on-fail passthrough
            primary
                script-url "cf3:/scripts/dyn_services.py"
                no shutdown
            exit
            secondary
                script-url ftp://*:*@10.255.137.80/scripts/dyn_services.py"
                no shutdown
            exit
     exit
exit
```

The parameters are no different to what have been defined generally for the use of Python scripting on the BNG.

When the very first session request arrives, the Python script is loaded into memory and executed. For all subsequent session requests the script is executed without the need for a reload. It is possible for both primary and secondary locations to be FTP sites (the small transfer delay for the first session is acceptable), however, it is recommended to have a compact-flash (cf1 or cf2) as the primary location and a remote location as backup.

Building Block: RADIUS Attributes

A series of Alcatel-Lucent vendor specific attributes (VSAs) have been defined to setup, teardown or modify dynamic data services from RADIUS.

The VSAs and their meaning are as follows:

- Alc-Dyn-Serv-SAP-Id [26-6527-164], type “string”

  This attribute identifies the dynamic service SAP. The format can be any valid Ethernet SAP format (dot1q or qinq encapsulation), including LAGs. A wildcard (“#”) can be specified for the port field and optionally for one of the tag fields of a qinq interface. To define the dynamic data service SAP-ID, the wildcard fields are replaced with the corresponding field from the Control Channel SAP-ID.

  Examples: “1/2/7:10.100” or “#:#.100”

- Alc-Dyn-Serv-Script-Action [26-6527-166], type “integer”

  A mandatory VSA in a COA to the control channel accounting session ID or the accounting session ID of the dynamic data service (only applicable for modify or teardown). Tells the system what script action is required: setup, modify or teardown of a dynamic data service.

  Values: 1=setup, 2=modify, 3=teardown

- Alc-Dyn-Serv-Policy [26-6527-167], type “string”

  Specifies the dynamic service policy to use for provisioning the dynamic service. The policy must be configured in the “configure service dynamic-services dynamic-services-policy < dynsrv-policy-name>” CLI context.

- Alc-Dyn-Serv-Script-Params [26-6527-165], type “string”

  This VSA contains parameters that can be used by the Python script to setup or modify a dynamic data service. The parameters can be split into multiple instances of the same attribute, linked together by the same tag, that is, the parameters can cross an attribute boundary. The concatenation of all “Alc-Dyn-Serv-Script-Params” attributes with the same tag in a single message must be formatted as “function-key = {dictionary}” where function-key specifies which Python functions will be called and {dictionary} contains the actual parameters in a Python dictionary structure format.

Example: “business_1 = { 'as_id' : '100', 'comm_id' : '200', 'if_name' : 'itf1', 'ipv4_address' : '172.16.1.1', 'egr_ip_filter' : '100' , 'routes' : [{'to' : '172.16.100.0/24', 'next-hop' : '172.16.1.2'}, {'to' : '172.16.200.0/24', 'next-hop' : '172.16.1.2'}]}

The above example shows each parameter with a keyword and the associated value. Alternatively only the parameter values can be sent with a pre-defined (and always constant) sequence.

Example: “business_1 = {“t”: '100', '200', 'itf1', '172.16.1.1', '100', '172.16.100.0/24', '172.16.1.2', '172.16.200.0/24', '172.16.1.2'}.

- Alc-Dyn-Serv-Acct-Interim-Ivl-1 [26-6527-168], type “integer”

  This VSA defines the number of seconds between each accounting interim update message for the primary accounting server. It overrides the local configured “update-interval” value in the dynamic services policy “accounting-1” CLI context. A value of 0 (zero) corresponds to no accounting interim update messages. A value [1..299] seconds is rounded to 300s (min. CLI value) and a value above 15552000 seconds (180 days, maximum CLI value) is rounded to the maximum CLI value.

  Range = 0 | [300 - 15552000].

- Alc-Dyn-Serv-Acct-Interim-Ivl-2 [26-6527-169], type “integer”

  Same function and values as Alc-Dyn-Serv-Acct-Interim-Ivl-1 [26-6527-168], for the second accounting server. It overrides the locally configured “update-interval” value in the dynamic services policy “accounting-2” CLI context.

- Alc-Dyn-Serv-Acct-Stats-Type-1 [26-6527-170], type “integer”

  Enable or disable dynamic data service accounting to the primary accounting server and specify the type of statistics that should be reported: volume and time or time only. It overrides the locally configured value in the dynamic services policy “accounting-1” CLI context.

  Values: 1=off, 2=volume-time, 3=time

- Alc-Dyn-Serv-Acct-Stats-Type-2 [26-6527-171], type “integer”

  Enable or disable dynamic data service accounting to the secondary accounting server and specify the type of statistics that should be reported: volume and time or time only. It overrides the locally configured “stats-type” value in the dynamic services policy “accounting-2” CLI context.

  Values: 1=off, 2=volume-time, 3=time

All VSAs are tagged to enable manipulation of up to 32 (tag values 0..31) dynamic data services in a single RADIUS message. VSAs with an identical tag belong to the same dynamic data service.

The use of the VSAs in RADIUS Access-Accept, CoA and Disconnect Messages is detailed in Table 43. An Access-Accept message can only contain dynamic data service setup requests. A CoA can be used to setup, modify or terminate a dynamic data service. A Disconnect Message can only be used to terminate a dynamic data service.

Table 43: Dynamic Service Attribute List for Setup, Modify and Teardown

Attribute Name	Access Accept	CoA			Disc. Message	Comment
	Setup	Setup	Modify	Tear-down	Tear-down	
Acct-Session-Id	N/A	M	M	M	M	Acct-Session-Id of the Control Channel or in case of a CoA: any other valid CoA key for ESM hosts/ sessions.
Alc-Dyn-Serv-SAP-Id	M	M(*)	M(*)	M(*)	N/A	Identifies the dynamic data service
Alc-Dyn-Serv-Script-Params	O	M(*)	M(*)	N/A	N/A	For a Modify, the script parameters represent the new parameters required for the change.
Alc-Dyn-Serv-Script-Action	O	M(*)	M(*)	M(*)	N/A	Must be “setup” if specified in an access-accept.
Alc-Dyn-Serv-Policy	O	O	O	O	N/A	The **default** policy used when not specified for create. In CoA, must be same as used for Setup if Specified for Modify or Teardown.
Alc-Dyn-Serv-Acct-Interim-Ivl-1	O	O	X(**)	X(**)	N/A	
Alc-Dyn-Serv-Acct-Interim-Iv2	O	O	X(**))	X(**))	N/A	
Alc-Dyn-Serv-Acct-Stats-Type-1	O	O	X(**)	X(**))	N/A	
Alc-Dyn-Serv-Acct-Stats-Type-2	O	O	X(**))	X(**))	N/A	

M = Mandatory, O= Optional, X = May not, N/A = Not Applicable (ignored)

(*) = CoA Nak’d, if not specified (Error Cause: 402 - Missing Attribute)

(**) = CoA Nak’d if specified (Error Cause:405 - Unsupported Service)

To summarize, Table 44 shows resulting dynamic service script actions as function of the RADIUS message (Access-Accept, CoA or DM) and the target (Control Channel or Dynamic Service SAP).

Table 44: Dynamic Service Actions on Control- and Data-Channel

Target	RADIUS Message	Dynamic Service Script Action	Comments
Control Channel	Access-Accept	Setup	Up to 32 dynamic data services in single message. Alc-Dyn-Serv-Script-Action VSA optional.
		Modify/Teardown	Not supported.
	CoA (acct-session-id or any other valid CoA key for ESM hosts/sessions)	Create/Modify/ Teardown	Cannot be mixed with session/post parameter changes in the same RADIUS message (results in CoA NAK). Up to 32 dynamic data services in single message. Alc-Dyn-Serv-Script-Action VSA mandatory.
	Disconnect	N/A	Teardown the Control Channel session and all associated dynamic data services.
Dynamic Service	CoA (acct-session-id of the dynamic data service sap)	Modify/Teardown	Only single dynamic data service per message (Acct-Session-Id). Alc-Dyn-Serv-Script-Action VSA mandatory.
		Setup	Not supported.
	Disconnect (acct-session-id of the dynamic data service sap)	N/A	Teardown the corresponding dynamic data service.

Building Block: Python Script

Dynamic data services scripts are built using a Python script engine. The following dedicated functions are available in the alc.dyn module:

- dyn.reference(function-key, reference-id string, dictionary)

  This function creates a dynamic reference to another function in the script. This function eases the creation of N:1 relationships in the script. For more information about use cases, see Building Block: CLI Snippets on page 2621. The function-key specifies the key in the action dictionary to find the corresponding setup/modify/teardown function calls.

  The reference-id (typically derived from a parameter specified from RADIUS, for example: service-name) specifies a unique instance string that identifies this reference.

  The dictionary specifies a dictionary with parameters that can be used in the parent function to generate CLI script output.

- dyn.action(d)

  When called, the dyn.action will take the “function-key” string specified in the Alc-Dyn-Serv-Script-Params attribute, and perform a lookup in the specified dictionary d to find the corresponding Python function to execute. The format of the dictionary is d = {key-1 : (Setup-1, Modify-1, Revert-1, Teardown-1), …, key-n : (Setup-n, Modify-n, Revert-n, Teardown-n) }. If the function-key matches, for example, key-1 and the corresponding Alc-Dyn-Script-Action is “setup”, then the function specified as “Setup-1” will be executed. Setup and teardown functions are mandatory. Modify and revert functions are optional. If a modify function is defined, a corresponding revert function must also be defined. If no modify/revert function is required, the keyword “None” should be used instead.

- dyn.add_cli(string)

  This function is used to generate CLI output in the Python script. The use of dyn.add_cli (”””) allows the specification of strings spanning multiple lines, which drastically improves the readability of the script.

  A subset of all available CLI commands is currently enabled for dynamic data services. The command “tools dump service dynamic-services command-list” provides a complete overview of all available CLI nodes for dynamic data services. In the allowed nodes section, all CLI nodes are listed that can be navigated to and where attributes can be modified. The pass through nodes section shows CLI nodes that can be navigated to but no attribute changes are allowed. For example, it is not allowed to change the autonomous system of a router (configure router autonomous-system <autonomous-system>) because “configure router” is a “pass through node”. However, you can navigate to configure router, because you can add a static route: “/configure router static-route 0.0.0.0/0 next-hop 192.168.1.1” is part of the “allowed nodes”.

- dyn.select_free_id(“service-id”)

  This function is used to select a free service ID within the service ID range defined under dynamic-services context. An automatic assignment of the service id is one option, but it is also possible to provide the service id as one of the parameters in the “Alc-Dyn-Serv-Script-Params” list from RADIUS.

  The service-ID is a node-internal attribute. As such it is valid to let the node select the ID itself. However, in a network with multiple BNGs and a single customer service spanning two or more BNGs, a network administrator may actually prefer to use the same service-id for this customer service on all nodes for better visibility, which cannot be guaranteed if the automatic option is chosen. 5620 SAM is also using the service-ID as one attribute in addition to others to discover service-entities across the whole network. If SAM is in use for general management and service assurance, it is advised to manually specify the service-ID and not to use the automatic selection.

  In any case, the administrator needs to make a choice between the automatic ID assignment and the specific assignment for all dynamic data services, as a mix between both is not recommended.

  When the automatic assignment is chosen, there is no “binding/memory” of a service ID to a provisioned service, which means a service that may have service ID xyz initially may get another service ID the next time it comes up. In other words, as soon as a service is disconnected, the service ID is freed up for the next activated service.

- dyn.get_sap()

  This function returns the value of the evaluation of the “Alc-Dyn-Serv-SAP-Id” attribute as a string. Wildcards (“#”) in the Alc-Dyn-Serv-SAP-Id are replaced with the corresponding port/vlan information of the control channel SAP-ID. So if, for example, the “Alc-Dyn-Serv-SAP-Id” contains “#:#.1” and the control channel SAP ID is “1/1/5:100.100”, the resulting SAP for the data service would be “1/1/5:100.1”.

- dyn.get_circuit_id()

  This function returns a string which is equal to the Control Channel Circuit-ID (from the DHCP relay agent option 82 or PPP tags). This function may be useful, for example, to use the circuit id in the SAP description.

- dyn.get_remote_id()

  This function returns a string which is equal to the Control Channel Remote-ID (from the DHCP relay agent option 82 or PPP tags). This function may be useful, for example, to use the remote id in the SAP description.

In addition to the RADIUS dictionary, the node will also store service-related parameters in a service-specific dictionary. The information in the RADIUS messages or in the stored dictionary are used for the various functions as outlined in Table 45:

Table 45: Function and Dictionary Relationship

Function Name	Input	Returns
setup dynsvc(rd*)	**rd:** radius dictionary in the parameter list in Alc-Dyn-Serv-Script-Params. VSA Passed to setup function.	A dictionary that will be stored for the lifetime of the dynamic service (sd).
modify_dynsvc(rd,sd**)	**rd:** radius dictionary in the parameter list in Alc-Dyn-Serv-Script-Params. VSA passed to modify function. **sd:** previously stored dictionary of the setup/previous modify functions.	Updated stored dictionary (sd)
revert_dynsvc(rd, sd)	**rd:** radius dictionary in the parameter list in Alc-Dyn-Serv-Script-Params. VSA passed to revert function. **sd:** previously stored dictionary of the setup/previous modify function.	The function does not return (store) any information. The previously stored dictionary (sd) is kept.
teardown_dynsvc(sd)	**sd:** previously stored dictionary by the setup function or a previous modify function are passed to the teardown function.	The function does not return (store) any information. The stored dictionary (sd) is deleted.

(*) rd = radius dictionary

(**) sd = stored dictionary. sd is required for modifies, reverts and teardowns.

Building Block: CLI Snippets

The necessary functional parts of a service configuration cannot typically be put into one large script (one single actionable function). This is best described with a small and simple example:

Imagine a single script where the setup action creates both the service instance and the SAP, and the teardown action removes the service instance and the SAP. For a service with just one SAP per service instance this may work fine, however, in a multi SAP service like a VPLS this will cause problems, especially during the service teardown action. This is because if multiple SAPs have been instantiated in a single service, the disconnect of just one SAP would trigger the teardown action which would try to remove the SAP (still ok) but then would try to remove the service instance. This action would fail as other SAPs still exist in the service. As such the script execution would fail.

It is therefore necessary to structure the whole required configuration into individual actionable pieces which are referenced by each other with specific reference-IDs. Those actionable pieces are called “snippets”.

Referenced snippets may or may not be executed depending on whether the functional instance exists already or not. As shown on the left of the picture below, the action to create a SAP references the creation of a service and then to the creation of a customer. For the very first business site to come up all three snippets will be executed. For any further business site to come up in the same service the script to create the SAP will be executed, the referenced service script and subsequently the customer script will not be executed again as those instances already exist. The same logic applies during the teardown action. Only when the last SAP in a service is removed is the service-instance itself removed, and potentially also the customer (unless it too is associated with other services).

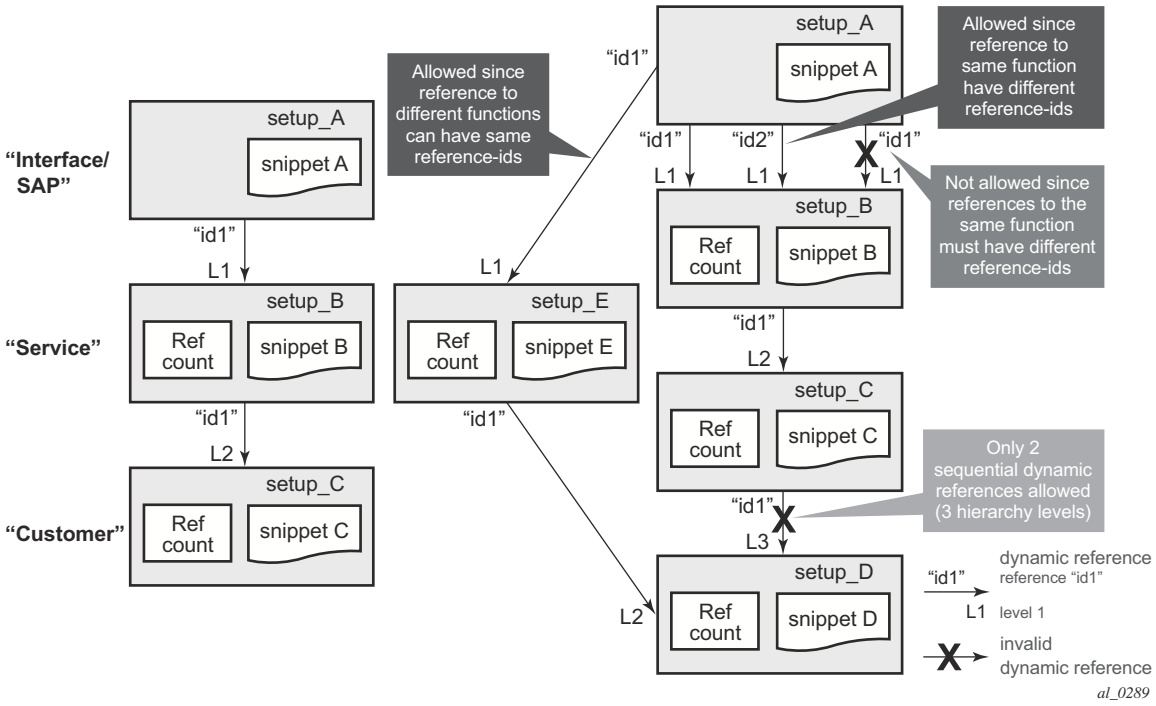

Figure 401: Hierarchy of Snippets

The implementation supports a three level hierarchy of snippets for high flexibility as shown in the picture. A reference to the fourth level as shown on the right side would result in an error.

Furthermore, snippets can be scaled "horizontally", so from one level multiple references to other snippets are possible. An example for that would be the creation of a SAP triggers the creation of a service as well as the creation of an Ethernet CFM association for that SAP.

Identifiers are needed for the referencing. The same identifier can be used on the "horizontal" level, but not on the vertical level between the same pair of snippets, also shown above.

Snippets are heavily used in the service examples in Bringing it all together on page 2624 where the logic and the referencing are described with real data.

Building Block: RADIUS Accounting

As dynamic data services are instantiated through RADIUS, it is also typically required to provide feedback to the RADIUS server for service establishment and teardown. This is achieved via RADIUS accounting records for the dynamic data channels in addition to the accounting messages for the PPP or DHCP control channel.

Up to two dedicated accounting destinations can be defined within the dynamic services policy. Thus, the accounting for the dynamic data services can be handled by an independent set of accounting servers (from the accounting for general ESM subscribers). But the same servers can also be used.

Each dynamic data service has its own accounting start/stop/interim messages based on a unique accounting session ID. In addition, the accounting packets contain a multi-session ID which is identical to the accounting session ID of the control channel and is therefore displayed in show commands as Acct-Session-ID-Ctrl as shown below.

```
A:BNG-1# show service dynamic-services saps summary
===============================================================================
Dynamic Services SAP's summary
===============================================================================
SAP                           Acct-Session-ID        Acct-Session-ID-Ctrl
-------------------------------------------------------------------------------
3/2/1:4.3                     D6E559000000B951668AEB D6E559000000B851668AEB
3/2/2:1.1                     D6E559000000C75166CFF4 D6E559000000C45166CFF4
3/2/2:1.2                     D6E559000000C85166CFF4 D6E559000000C45166CFF4
3/2/2:1.3                     D6E559000000C95166CFF4 D6E559000000C45166CFF4
3/2/2:1.4                     D6E559000000CA5166CFF4 D6E559000000C45166CFF4
-------------------------------------------------------------------------------
No. of SAP's: 5
===============================================================================
```

The Accounting Session ID (in the centre above) is the one for the dynamic data service itself, the one on the right is from the control-channel. The above example clearly shows that the last 4 dynamic services all belong to the same control channel, as they all have the same Acct-Session-ID-Ctrl.

If the accounting stats-type is set to “volume-time”, the interim and stop accounting messages will also contain counters for the data traffic through the service. With the accounting stats-type “time”, no counters are included, only session time is reported.

As a dynamic data service is functionally no different from a regular data service, traffic volumes can also be gathered by assigning accounting policies within the service for file-based XML accounting.

Bringing it all together

This section gives examples of all of the above parameters and will also cover show, log and debug information.

In the given example, a single user in the database has four different associated data services. Not only are the data service types all different, but also other aspects of the parameter set, this has an effect on how the data is entered in the RADIUS VSAs and how the Python script is constructed. More detail is given below. The different models for specifying parameters are presented to show the flexibility. An operator typically chooses a single model and uses that for all its services.

As all of the information for these four services will potentially be sent in one RADIUS message, the VSAs need to be tagged so that the BNG can link the appropriate VSAs to each other and differentiate the services. For better visibility, the different sections in the RADIUS users file are displayed with bold black and dark grey text.

The freeradius users file format is used for this example.

```
1.   "subscriber12@domain2.com" Cleartext-Password := "ALU"
2.        Alc-Subsc-ID-Str := "pppoe-user12",
3.        Framed-IP-Address = 10.2.1.200,
4         Alc-Dyn-Serv-SAP-Id:1  = "#:#.1",
5.        Alc-Dyn-Serv-Script-Params:1 = "business_epipe={'t':('EPipe-
6.           CustomerName','CustomerName-Circuit-1','3','3','64496',
7.           '192.0.2.5','192.0.2.1','3333')}",
8.        Alc-Dyn-Serv-Policy:1 =  "dynamic-services-2",
9.        Alc-Dyn-Serv-SAP-Id:2 += "#:#.2",
10.       Alc-Dyn-Serv-Script-Params:2 += "business_vprn={'t':('9999',
11.          'VPRN-CustomerName','64497','100000','CustomerName-Circuit-
12.          1','172.16.10.1/30','3','1','3','2','172.16.100.0/24',
13.          '172.16.10.2','100')}",
14.       Alc-Dyn-Serv-Acct-Interim-Ivl-1:2 += "600",
15.       Alc-Dyn-Serv-Acct-Interim-Ivl-2:2 += "0",
16.       Alc-Dyn-Serv-Policy:2  +=  "dynamic-services-2",
17.       Alc-Dyn-Serv-SAP-Id:3  += "#:#.3",
18.       Alc-Dyn-Serv-Script-Params:3 += "business_vpls={'inst':
19.          'VPLS-CustomerName','if_name':'CustomerName-Circuit-1','ing_qos':'3',
20.          'egr_qos':'3','imp_comm_val':'10000','exp_comm_val':'10000',
21.          'rt':'64498','rd':'64498'}",
22.       Alc-Dyn-Serv-Policy:3 +=  "dynamic-services-2",
23.       Alc-Dyn-Serv-Acct-Interim-Ivl-1:3 += "0",
24.       Alc-Dyn-Serv-Acct-Interim-Ivl-2:3 += "0",
25.       Alc-Dyn-Serv-Acct-Stats-Type-1:3 += off,
26.       Alc-Dyn-Serv-Acct-Stats-Type-2:3 += off,
27.       Alc-Dyn-Serv-SAP-Id:4  += "#:#.4",
28.       Alc-Dyn-Serv-Script-Params:4 += "business_ies={'t':
29.          ('IES-CustomerName','CustomerName-Circuit-1','172.16.11.1/30',
30.          '2001:db8:5100:1000::1/64','5','1','1','6','2','2','5','25',
31.          'cfm-Mep-to-CPE','100',",
32.       Alc-Dyn-Serv-Script-Params:4 += "[{'to':'172.16.110.0/24',
33,          'n-h':'172.16.11.2'},{'to':'2001:db8:bbbb::/56',
34.          'n-h':'2001:db8:5100:1000::2'}])}",
35.       Alc-Dyn-Serv-Policy:4 +=  "dynamic-services-2",
36.       Alc-Dyn-Serv-Acct-Interim-Ivl-1:4 += "600",
37.       Alc-Dyn-Serv-Acct-Interim-Ivl-2:4 += "0",
38.       Alc-Dyn-Serv-Acct-Stats-Type-1:4 += "3",
39.       Alc-Dyn-Serv-Acct-Stats-Type-2:4 += "2",
```

The first section (lines 1 — 3) shows a minimal parameter set for the (PPP) control channel. As the focus of this example is on the dynamic data services, all default parameters will be used for the control-session which are defined under the msap.

The second section (lines 4 — 8, attributes with tag ":1") shows a possible parameter set for an Epipe service. Only the absolutely minimum set of VSAs is used (see Table 43, Dynamic Service Attribute List for Setup, Modify and Teardown, on page 2616). Furthermore, all service parameters are listed without keywords in a pre-defined order. No service ID number is specified in "Alc-Dyn-Serv-Script-Params", hence the Python script should dynamically select the next free ID.

The third section (lines 9 — 16, attributes with tag ":2") shows a possible parameter set for a VPRN service. A few more VSAs are defined, thus some of the default parameters in the dynamic service policy are overwritten for this service. The first entry in the "Alc-Dyn-Serv-Script-Params" attribute specifies the Service-ID number for this service, so the Python script should not select a service ID automatically. Furthermore, static-routing information towards the CPE is added as normal attributes at the end of the list.

The fourth section (line 17 — 26, attributes with tag ":3") shows a possible parameter set for a VPLS service. Notice the difference with the first two services in the "Alc-Dyn-Serv-Script-Params" part: now all parameters are given their specific keyword. As such, the sequence of those parameters is not important. The effect on the Python script is shown further down.

The fifth section (lines 27 — 39, attributes with tag ":4") finally shows a possible parameter set for an IES service. All of the required parameters for this service do not fit into a single "Alc-Dyn-Serv-Script-Params" attribute anymore (limited to 247 bytes). As is shown, multiple VSAs can be "concatenated" by simply splitting the attributes. It is important that the order in which the different "Alc-Dyn-Serv-Script-Params" attributes with the same tag is received can be guaranteed. Furthermore the second appearance of this VSA shows a different way of provisioning static-routing information towards the CPE.

To better understand the details it is necessary to take a closer look into the active Python script. The first important part is the section with the dynamic actions.

```
-snip-

d = {
"vprn": (setup_vprn, None, None, teardown_vprn),
"ies": (setup_ies, None, None, teardown_ies),
"vpls": (setup_vpls, None, None, teardown_vpls),
"epipe": (setup_epipe, None, None, teardown_epipe),
"ethcfm" : (setup_ethcfm_domain, None, None, teardown_ethcfm_domain),
"business_vprn" : (setup_business_vprn, None, None, teardown_business_vprn),
"business_ies" : (setup_business_ies, None, None, teardown_business_ies),
"business_vpls" : (setup_business_vpls, None, None, teardown_business_vpls),
"business_epipe" : (setup_business_epipe, modify_business_epipe,
                    revert_business_epipe, teardown_business_epipe)}
dyn.action(d)
```

The function-key string specified at the start of the "Alc-Dyn-Serv-Script-Params" (for example Alc-Dyn-Serv-Script-Params:1 = "business_epipe={…}") has a 1:1 mapping with the keys of the dictionary "d" in the highlighted section of the above sample (for example d = { …, "business_epipe" : (…)}. For services, different values for setup, modify, revert and teardown are given which point to other sections in the Python script (see below). Setup and teardown functions are mandatory, whereas modify and revert functions are optional.

In the unbolded text of the previous example, there are other actions defined that are not contained in the RADIUS attributes (for example d = { "vprn": (…) , …). Those actions are referenced by the four main functions.

In the next part, there is more detail presented in each service example and maps it to the corresponding Python function.

It is advisable to read through all examples, as only the deltas between each service are explicitly explained.

Example 1 – Epipe service

```
# copy of the RADIUS attributes from above
-snip-
        Alc-Dyn-Serv-SAP-Id:1  = "#:#.1",
        Alc-Dyn-Serv-Script-Params:1 = "business_epipe={'t':
            ('EPipe-CustomerName','CustomerName-Circuit-1','3','3','64496'
             ,'192.0.2.5','192.0.2.1','3333')}",
        Alc-Dyn-Serv-Policy:1 =  "dynamic-services-2",
-snip-

# Python-part
d = {
-snip-
"business_epipe" : (setup_business_epipe, modify_business_epipe,
                    revert_business_epipe, teardown_business_epipe)
dyn.action(d)
-snip-
def setup_business_epipe(d):
    keys = ('inst', 'if_name', 'ing_qos', 'egr_qos', 'as', 'remote_ip',
           'local_ip', 'glb_svc_id')
    d = dict(zip(keys, d['t']))
    ref_d = dyn.reference("epipe", d['inst'], d)
    d['svc_id'] = ref_d['svc_id']
    d['sap_id'] = dyn.get_sap()
    dyn.add_cli("""
configure
  service
    epipe %(svc_id)s
      sap %(sap_id)s create
        description "%(if_name)s"
        ingress
          qos %(ing_qos)s
        exit
        egress
          qos %(egr_qos)s
        exit
      exit
      spoke-sdp-fec %(svc_id)s fec 129 aii-type 2 create
        pw-template-bind 2
        saii-type2 %(as)s:%(local_ip)s:%(glb_svc_id)s
        taii-type2 %(as)s:%(remote_ip)s:%(glb_svc_id)s
        no shutdown
      exit
    exit
  exit
exit
""" % d)
    return d

def setup_epipe(d):
```

```
    d['svc_id'] = dyn.select_free_id("service-id")
    dyn.add_cli("""
configure
  service
    epipe %(svc_id)s customer 1 create
      service-name "%(inst)s"
      description "%(inst)s"
      no shutdown
    exit
  exit
exit
""" % d)
    return {'svc_id':d['svc_id']}

def teardown_epipe(d):
    dyn.add_cli("""
configure
  service
    epipe %(svc_id)s
      shutdown
    exit
    no epipe %(svc_id)s
  exit
exit
""" % d)

def teardown_business_epipe(d):
    dyn.add_cli("""
configure
  service
    epipe %(svc_id)s
      sap %(sap_id)s
        shutdown
      exit
      spoke-sdp-fec %(svc_id)s
        shutdown
      exit
      no sap %(sap_id)s
      no spoke-sdp-fec %(svc_id)s
    exit
  exit
exit
""" % d)
-snip-
```

Based on the dictionary specified in the dyn.action(d) call, the function definition “setup_business_epipe” in the Python script corresponds with the function that will be called if the function-key “business-epipe” is specified in the “Alc-Dyn-Serv-Script-Params” attribute as dictionary name and if a setup action is required. The dictionary containing the parameters in the RADIUS VSA “Alc-Dyn-Serv-Script-Params” has a single key-value pair, with the parameters stored in a tuple. The individual parameters cannot be identified with a keyword hence the order in which they are specified in the RADIUS VSA should match the order in which they are extracted in the Python script. The first two lines in this part of the script extract the parameters out of the array “t” and link them to unique keywords, which are used for the rest of the script.

The parameter “inst” is important in this logic, as it defines whether access circuits belong to the same service-instance or different instances (the RADIUS VSAs for two SAPs belonging to the same service therefore need to have the same “inst” value). If you look at the CLI of the “setup_business_epipe” function, you can see that it creates the SAP and all related attributes, but not the service itself. It is the “ref_d = dyn.reference("epipe", d['inst'], d)” that references a part in the script to create the actual service-instance. The referenced function is found by using the first parameter in the dyn.reference call (“epipe”) as a function-key lookup in the dictionary specified in the dyn.action(d) and finding the corresponding setup function: d = { …, "epipe" : (setup_epipe, …), …}. The second parameter (“d['inst']”) is used as unique identification of the service instance. The last parameter (“d”) is a dictionary with parameters that can be used by the references function. When the first customer endpoint with a new “inst” name comes up, the service itself gets created.

By looking at “def setup_epipe(d):” the first line “d['svc_id'] = dyn.select_free_id("service-id")” of the script automatically picks a free service-id out of the range defined in the dynamic service policy, as no service ID was provided in the RADIUS parameters. The rest of this function creates the service instance. Service attributes that were provided by RADIUS and are placed in a service specific dictionary are available to this function via the third parameter in the dyn.reference call. The newly generated service ID is returned to the calling script by the “return {'svc_id':d['svc_id']}” command at the end of the function. The service specific dictionary (as explained in the Python Script Building Block) is updated with the appropriate information.

Back to “def setup_business_epipe(d):”, the service ID together with the SAP ID and the parameters from the Alc-Dyn-Serv-Script-Params VSA are used to create the appropriate CLI code for the SAP and the SDP within the service.

Similar to the setup, there is also a teardown part for both service and SAP. The teardown function is called either through the termination of the control-channel, through a COA with Alc-Dyn-Script-Action = teardown or through a disconnect message. The CLI for the teardown script must be written in the correct sequence as applied by the SR OS CLI logic so that SAP(s) and service(s) are removed in the correct order.

Example 2 – VPRN service

```
RADIUS-part from above
-snip-
        Alc-Dyn-Serv-SAP-Id:2 += "#:#.2",
        Alc-Dyn-Serv-Script-Params:2 += "business_vprn={'t':
            ('9999','VPRN-CustomerName','64497','100000',
             'CustomerName-Circuit-1','172.16.10.1/30','3','1','3','2',
             '172.16.100.0/24','172.16.10.2','100')}",
        Alc-Dyn-Serv-Acct-Interim-Ivl-1:2 += "600",
        Alc-Dyn-Serv-Acct-Interim-Ivl-2:2 += "0",
        Alc-Dyn-Serv-Policy:2  +=  "dynamic-services-2",
-snip-

Python-part
d = {
-snip-
"business_vprn" : (setup_business_vprn, None, None, teardown_business_vprn)
```

```
dyn.action(d)
-snip-
def setup_business_vprn(d):
    keys = ('svc_id', 'inst', 'as_id', 'comm_id', 'if_name', 'ipv4_address',
            'ing_qos', 'ing_ip_filter', 'egr_qos', 'egr_ip_filter', 'lan_pfx',
            'nxt_hop', 'metric')
    d = dict(zip(keys, d['t']))
    ref_d = dyn.reference("vprn", d['inst'], d)
    d['sap_id'] = dyn.get_sap()
    dyn.add_cli("""
configure
  service
    vprn %(svc_id)s
      interface "%(if_name)s" create
        address %(ipv4_address)s
        urpf-check mode strict
        sap %(sap_id)s create
          ingress
            qos %(ing_qos)s
            filter ip %(ing_ip_filter)s
          exit
          egress
            qos %(egr_qos)s
            filter ip %(egr_ip_filter)s
          exit
        exit
      exit
    exit
  exit
  router
    static-route %(lan_pfx)s next-hop %(nxt_hop)s metric %(metric)s
  exit
exit
""" % d)
    return d

def setup_vprn(d):
    dyn.add_cli("""
configure
  service
    vprn %(svc_id)s customer 1 create
      service-name "%(inst)s"
      description "%(inst)s"
      autonomous-system %(as_id)s
      route-distinguisher %(as_id)s:%(comm_id)s
      auto-bind mpls
      vrf-target target:%(as_id)s:%(comm_id)s
      no shutdown
    exit
  exit
exit
""" % d)
    return {'svc_id':d['svc_id']}

def teardown_vprn(d):
    dyn.add_cli("""
configure
  service
    vprn %(svc_id)s
```

```
      shutdown
    exit
    no vprn %(svc_id)s
  exit
exit
""" % d)

def teardown_business_vprn(d):
    dyn.add_cli("""
configure
   router
     no static-route %(lan_pfx)s next-hop %(nxt_hop)s
   exit
   service
     vprn %(svc_id)s
       interface "%(if_name)s"
         sap %(sap_id)s
           shutdown
         exit
         no sap %(sap_id)s
         shutdown
       exit
       no interface "%(if_name)s"
     exit
   exit
exit
""" % d)
-snip-
```

In this example of a VPRN service two additional RADIUS VSAs are used to overwrite the accounting interim update intervals for the two RADIUS Accounting servers that are specified in the dynamic services policy. The Stats-Type configuration (time or volume-time) is obtained from the dynamic services policy as no RADIUS VSA is provided for that.

The beginning of the “setup_business_vprn” definition is identical to the earlier Epipe service example. This time a service identifier is provided as part of the parameter list. The referenced function to create the VPRN service (def setup_vprn) does not need the line to auto-generate the service ID.

At the end of the setup-procedure there is a basic example to add static-route information in case they are needed for PE-CE communication. Later on, in the IES service example, a more flexible alternative is shown.

Example 3 – VPLS service

```
RADIUS-part from above
-snip-
        Alc-Dyn-Serv-SAP-Id:3  += "#:#.3",
        Alc-Dyn-Serv-Script-Params:3 += "business_vpls={'inst':
            'VPLS-CustomerName','if_name':'CustomerName-Circuit-1','ing_qos':'3',
            'egr_qos':'3','imp_comm_val':'10000','exp_comm_val':'10000',
            'rt':'64498','rd':'64498'}",
```

```
      Alc-Dyn-Serv-Policy:3 +=  "dynamic-services-2",
        Alc-Dyn-Serv-Acct-Interim-Ivl-1:3 += "0",
        Alc-Dyn-Serv-Acct-Interim-Ivl-2:3 += "0",
        Alc-Dyn-Serv-Acct-Stats-Type-1:3 += off,
        Alc-Dyn-Serv-Acct-Stats-Type-2:3 += off,
-snip-

Python-part
d = {
-snip-
"business_vpls" : (setup_business_vpls, None, None, teardown_business_vpls)
-snip-
def setup_business_vpls(d):
    ref_d = dyn.reference("vpls", d['inst'], d)
    d['svc_id'] = ref_d['svc_id']
    d['sap_id'] = dyn.get_sap()
    dyn.add_cli("""
configure
   service
    vpls %(svc_id)s
      sap %(sap_id)s create
        description "%(if_name)s"
        ingress
          qos %(ing_qos)s
        exit
        egress
          qos %(egr_qos)s
        exit
        collect-stats
        accounting-policy 10
      exit
    exit
  exit
exit
""" % d)
    return d

def setup_vpls(d):
    d['svc_id'] = dyn.select_free_id("service-id")
    dyn.add_cli("""
configure
  service
    vpls %(svc_id)s customer 1 create
      service-name "%(inst)s"
      description "%(inst)s"
      bgp
        route-distinguisher %(rd)s:%(exp_comm_val)s
        route-target export target:%(rt)s:%(exp_comm_val)s
                      import target:%(rt)s:%(imp_comm_val)s
        pw-template-binding 1
        exit
      exit
      bgp-ad
        vpls-id %(rt)s:%(exp_comm_val)s
        no shutdown
      exit
      no shutdown
    exit
  exit
```

```
exit
""" % d)
    return {'svc_id':d['svc_id']}

def teardown_vpls(d):
    dyn.add_cli("""
configure
  service
    vpls %(svc_id)s
      shutdown
      bgp-ad
        shutdown
      exit
      no bgp-ad
      bgp
        no pw-template-binding 1
      exit
    exit
    no vpls %(svc_id)s
  exit
exit
""" % d)

def teardown_business_vpls(d):
    dyn.add_cli("""
configure
  service
    vpls %(svc_id)s
      sap %(sap_id)s
        shutdown
      exit
      no sap %(sap_id)s
    exit
  exit
exit
""" % d)
-snip-
```

In the VPLS example the “Alc-Dyn-Serv-Acct-Stats-Type” is set to “off” for both RADIUS accounting destinations, meaning RADIUS accounting is switched off, even if it is enabled in the dynamic data services policy. In the script you can see that this service uses XML-accounting on the SAP instead (“collect-stats” and “accounting-policy 10”).

The dictionary containing the parameters in the RADIUS VSA “Alc-Dyn-Serv-Script-Params” has a key-value pair for each parameter. In the Python script the individual parameters can be identified immediately with the dictionary key. The order in which they are specified in the RADIUS VSA does not have to be strictly defined. The drawback of this approach is that the length of the parameter VSA increases. A single parameter VSA is limited to a length of 246 bytes and the total length of all parameter VSAs for a single service is limited to 1000 bytes.

Example 4 – IES service

```
RADIUS-part from above
-snip-
```

```
        Alc-Dyn-Serv-SAP-Id:4  += "#:#.4",
        Alc-Dyn-Serv-Script-Params:4 += "business_ies={'t':
            ('IES-CustomerName','CustomerName-Circuit-1','172.16.11.1/30',
             '2001:db8:5100:1000::1/64','5','1','1','6','2','2','5','25',
             'cfm-Mep-to-CPE','100',",
        Alc-Dyn-Serv-Script-Params:4 += "[{'to':'172.16.110.0/24',
            'n-h':'172.16.11.2'},{'to':'2001:db8:bbbb::/56',
            'n-h':'2001:db8:5100:1000::2'}])}",
        Alc-Dyn-Serv-Policy:4 +=  "dynamic-services-2",
        Alc-Dyn-Serv-Acct-Interim-Ivl-1:4 += "600",
        Alc-Dyn-Serv-Acct-Interim-Ivl-2:4 += "0",
        Alc-Dyn-Serv-Acct-Stats-Type-1:4 += "3",
        Alc-Dyn-Serv-Acct-Stats-Type-2:4 += "2",
-snip-

Python-part
d = {
-snip-
"business_ies" : (setup_business_ies, None, None, teardown_business_ies)
-snip-
def setup_business_ies(d):
    keys = ('inst', 'if_name', 'ipv4_address', 'ipv6_address', 'ing_qos',
            'ing_ip_filter', 'ing_ipv6_filter',  'egr_qos', 'egr_ip_filter',
            'egr_ipv6_filter', 'ing_bw', 'egr_bw', 'cfm_assoc_id', 'metric',
            'routes')
    d = dict(zip(keys, d['t']))
    ref_d = dyn.reference("ies", d['inst'], d)
    d['svc_id'] = ref_d['svc_id']
    d['sap_id'] = dyn.get_sap()
    d['cfm_domain'] = 1
    ref_d_cfm = dyn.reference("ethcfm", str(d['cfm_domain']), d)
    dyn.add_cli("""
configure
  eth-cfm
    domain %(cfm_domain)s
      association %(svc_id)s format string name "%(cfm_assoc_id)s"
        bridge-identifier %(svc_id)s
        exit
        ccm-interval 1
        remote-mepid 2
      exit
    exit
  exit
  service
    ies %(svc_id)s
      interface "%(if_name)s" create
        address %(ipv4_address)s
        urpf-check mode strict
        cflowd interface both
        ipv6
          address %(ipv6_address)s
          urpf-check mode strict
        exit
        sap %(sap_id)s create
           description "%(if_name)s"
          ingress
            scheduler-policy "Business Services"
            scheduler-override
               scheduler "root-t1" create
```

```
                rate %(ing_bw)s000
              exit
            exit
            qos %(ing_qos)s
            filter ip %(ing_ip_filter)s
            filter ipv6 %(ing_ipv6_filter)s
          exit
          egress
            qos %(egr_qos)s
            filter ip %(egr_ip_filter)s
            filter ipv6 %(egr_ip_filter)s
            agg-rate-limit %(egr_bw)s000 queue-frame-based-accounting
          exit
          collect-stats
          accounting-policy 10
          eth-cfm
            mep 1 domain %(cfm_domain)s association %(svc_id)s direction down
              ccm-enable
              no shutdown
            exit
          exit
        exit
        urpf-check
        exit
      exit
    exit
  exit
  router
""" % d)
    for route in d['routes']:
        dyn.add_cli("""
        static-route %s next-hop %s metric %s tag 80
"""% (route["to"], route["n-h"], d['metric']))
    dyn.add_cli("""
    exit
exit
""" % d)
    return d

def setup_ies(d):
    d['svc_id'] = dyn.select_free_id("service-id")
    dyn.add_cli("""
configure
  service
    ies %(svc_id)s customer 1 create
      service-name "%(inst)s"
      description "%(inst)s"
      no shutdown
    exit
  exit
exit
""" % d)
    return {'svc_id':d['svc_id']}

def setup_ethcfm_domain(d):
    dyn.add_cli("""
configure
  eth-cfm
    domain %(cfm_domain)s format none level 1
```

```
    exit
  exit
exit
""" % d)
    return {'cfm_domain':d['cfm_domain']}

def teardown_ethcfm_domain(d):
    dyn.add_cli("""
configure
  eth-cfm
    no domain %(cfm_domain)s
  exit
exit
""" % d)

def teardown_ies(d):
    dyn.add_cli("""
configure
  service
    ies %(svc_id)s
      shutdown
    exit
    no ies %(svc_id)s
  exit
exit
""" % d)

def teardown_business_ies(d):
    dyn.add_cli("""
configure
  router
""")
    for route in d['routes']:
        dyn.add_cli("""
        no static-route %s next-hop %s
"""% (route["to"], route["n-h"]))
    dyn.add_cli("""
   exit
exit
""")
    dyn.add_cli("""
configure
  service
    ies %(svc_id)s
      interface "%(if_name)s"
        sap %(sap_id)s
          shutdown
          eth-cfm
            mep 1 domain %(cfm_domain)s association %(svc_id)s
              shutdown
            exit
            no mep 1 domain %(cfm_domain)s association %(svc_id)s
          exit
        exit
        no sap %(sap_id)s
        shutdown
      exit
      no interface "%(if_name)s"
    exit
```

```
  exit
  eth-cfm
    domain %(cfm_domain)s
      association %(svc_id)s
        no bridge-identifier %(svc_id)s
      exit
      no association %(svc_id)s
    exit
  exit
exit
""" % d)
-snip-
```

The IES example has the most attributes. The maximum length of a tagged RADIUS VSA is 246 bytes. If the amount of data is too big to fit into one attribute, simply add a second or third one in the syntax shown above in the RADIUS part. There is no need to separate the attributes exactly at 246 bytes; it can be cut at any position in the list (preferably between two attributes for better readability). Note also that all the parameter VSAs that belong to the same service should have the same tag (":4" in this example).

In case of multiple parameter VSAs, the order in which they are specified is important and must be guaranteed as the concatenation of all the attributes must result in a Python dictionary in the form: "dictionary-name = {…}". The Python script is not aware that multiple attributes were used.

Another difference to the previous examples is that there is not only a reference to the function for the service creation, but also a similar reference to a function for Ethernet Connectivity Fault Management (CFM). Considering that you may want to put all of the Eth-CFM endpoints under the same domain within unique associations, the Eth-CFM domain needs to be created first and torn down as last.

Finally, a different way to provide static-route information is shown at the end of the "setup_business_ies" definition (starting with "for route in d['routes']:"). Also note the difference in how this information is implemented at the end of the "Alc-Dyn-Serv-Script-Params" list. The static routes themselves are defined as a dictionary and thus as many routes as required can be added with this method. Compare this to the VPRN example where a more basic mechanism was used.

As outlined before, dynamic data services can be triggered during the Access-Accept for the control channel but also through a CoA to the control channel Accounting Session ID.

Example 5 – modify an Epipe service using CoA

So far the focus was on service establishment and teardown. It is also possible to change a running dynamic data service using the "modify" function. This will be explained with the previously configured Epipe service.

```
RADIUS attributes in the COA message
```

```
   Acct-Session-Id = D6E559000000BD5166BF34 #
   Alc-Dyn-Serv-SAP-Id:1  = "#:#.1",
   Alc-Dyn-Serv-Script-Params:1 = "business_epipe={'ing_qos':'4','egr_qos':'4'}",
   Alc-Dyn-Serv-Script-Action:1 = modify,
   Alc-Dyn-Serv-Policy:1 =  "dynamic-services-2",


Python-part
d = {
-snip-
"business_epipe" : (setup_business_epipe, modify_business_epipe, revert_business_epipe,
teardown_business_epipe)}
dyn.action(d)
-snip-
def modify_business_epipe(d, sd):
    sd['ing_qos'] = d['ing_qos']
    sd['egr_qos'] = d['egr_qos']
    dyn.add_cli("""
configure
   service
      epipe %(svc_id)s
         sap %(sap_id)s
           ingress
              qos %(ing_qos)s
           exit
           egress
              qos %(egr_qos)s
           exit
         exit
       exit
    exit
exit
"""% sd)
    return sd

def revert_business_epipe(d, sd):
    dyn.add_cli("""
configure
   service
     epipe %(svc_id)s
        sap %(sap_id)s
          ingress
            qos %(ing_qos)s
          exit
          egress
            qos %(egr_qos)s
          exit
        exit
      exit
    exit
exit
""" % sd)
-snip-
```

Through the function-key in the parameter list (Alc-Dyn-Serv-Script-Params:1 = "business_epipe= …) and the action attribute of “modify” (Alc-Dyn-Serv-Script-Action:1 = modify), the script will identify the relevant routine to be invoked for the modification

(modify_business_epipe). If a modify function is defined, there must also be a definition for a revert function. A revert function cannot be initiated from RADIUS, but it is automatically executed to restore the initial configuration in case the modify script execution fails.

A modify action for an existing service is triggered with a CoA message. For this CoA, either the Accounting Session ID (ASID) of the control channel or the Accounting Session ID of the dynamic data channel can be used. In case the ASID of the control channel is used, the "Alc-Dyn-Serv-SAP-Id" can contain wildcards, as the appropriate port and VLAN information will be taken from the control channel. If the ASID of the dynamic data channel itself is used, the "Alc-Dyn-Serv-SAP-Id" needs to be fully specified, without wildcards. Otherwise the script execution will fail.

For a modify action, the "Alc-Dyn-Serv-Script-Params" only contains the parameters to be changed and does not need any further service identifying information. The service is identified based on the ASID and the "Alc-Dyn-Serv-SAP-Id". Parameters which have been previously received by the setup or an earlier modify function are available in the stored dictionary (sd). Those are combined with the dictionary in the RADIUS message (d). Service modifications which relate to subsequent modifications, or for the service teardown, need to be updated in the stored dictionary so that they can be used in those later actions. This is achieved by the "return sd" command.

As with "manual" provisioned services, the new QoS settings from our example take effect immediately.

A dynamic data service can also be disconnected using a RADIUS Disconnect Message containing the Accounting Session ID of the dynamic data service, or indirectly via a RADIUS Disconnect Message containing the Accounting Session ID of the control channel which would result in a teardown of all associated dynamic data services.

Debugging

It is obvious that the Python scripts need extensive testing in the lab before they are deployed in the field. This testing may require a number of iterations: write the script, testing, verification, improvement and testing again. Every time there is a change in the Python script the node needs to reload the script. This is achieved by a **shutdown** and **no shutdown** of the active script using the command:

configure aaa radius-script-policy <script-policy-name> <primary/secondary> shutdown
configure aaa radius-script-policy <script-policy-name> <primary/secondary> no shutdown

Testing the script may result in some problems if certain aspects may not work as expected (see also debug functions later in this section). It can be that a dynamically created service cannot be removed properly because the teardown script contains errors and the whole service, or fragments of that service, may still exist on the node.

Dynamic data services cannot be edited in normal CLI mode as it may potentially make a later removal of that service through the script impossible. For troubleshooting there is a procedure to

manipulate those services during the testing phase, thus avoiding the need to reboot the box to clear the state. The **enable-dynamic-services-config** command allows for the editing dynamic services just like normal services. As this is an action that should only be executed by authorized personnel, the activation of this command is protected by the use of a password, defined under **configure system security password dynsvc-password**.

The **show users** command has been extended to visualize the respective mode ('D' indicates a user is in dynamic service edit mode). A user in dynamic services edit mode cannot modify regular services.

no enable-dynamic-services-config returns the user to normal mode.

To support the creation and the troubleshooting during the test phase the SR OS debug functions have been extended extensively to allow for a detailed review of what is happening in the script and on the CLI.

```
debug dynamic-services
debug dynamic-services scripts
debug dynamic-services scripts event
debug dynamic-services scripts event cli
debug dynamic-services scripts event errors
debug dynamic-services scripts event executed-cmd
debug dynamic-services scripts event state-change
debug dynamic-services scripts event warnings
debug dynamic-services scripts instance
debug dynamic-services scripts instance event
debug dynamic-services scripts instance event cli
debug dynamic-services scripts instance event errors
debug dynamic-services scripts instance event executed-cmd
debug dynamic-services scripts instance event state-change
debug dynamic-services scripts instance event warnings
debug dynamic-services scripts script
debug dynamic-services scripts script event
debug dynamic-services scripts script event cli
debug dynamic-services scripts script event errors
debug dynamic-services scripts script event executed-cmd
debug dynamic-services scripts script event state-change
debug dynamic-services scripts script event warnings
```

It is advised to enable all debug options when starting and then remove more and more debugs options as the script becomes more complete and stable. The debug output gives clear indications about errors in the script or its execution in case something goes wrong.

An additional aid is the use of “print” commands in the Python script itself for certain attributes during the execution of the script. The print output will appear in the debug log. “Print” commands in the Python script should only be used during the testing phase and not in the normal operations mode.

The following command allows the execution of a dynamic services Python script without the need for RADIUS interaction:

tools perform service dynamic-services evaluate-script sap <sap-id> control-session <acct-session-id> action <script-action> [dynsvc-policy <name>]

show service dynamic-services script statistics provides general statistics about script execution.

show service dynamic-services script snippets displays the individual service configuration parts and allows to check if all “snippets” are actually referenced (the counter will increment/decrement with every function call).

In the case of a failed script action a SAP may not be deleted properly and it remains in the configuration as an “orphaned” object. “show service dynamic-services saps orphaned” displays orphaned objects.

An orphaned object no longer has any references, which can be seen using **show service dynamic-services root-objects** where the snippet name and snippet instance is set to “N/A”.

Complete setup flow example

To finalize the section about the interaction between RADIUS and the Python script, the complete setup flow for the Epipe example is shown using extracts from the debug output (any missing sequence numbers in the flow below are simple acknowledge messages from RADIUS and are left out to focus on the important information). The debug settings to be used for this output are the following.

```
*A:BNG-1# show debug
debug
    router "Base"
        radius
            packet-type authentication accounting coa
            detail-level medium
        exit
    exit
    router "management"
        radius
            packet-type authentication accounting coa
            detail-level medium
        exit
    exit
    dynamic-services
        scripts
            event
                cli
            exit
            instance "dynamic-services-1"
                event
                    cli
                exit
            exit
        exit
    exit
exit
```

The first sequence in the flow is the Access-Request to the RADIUS server for the control channel. The information provided is that configured as part of the regular ESM configuration.

```
9 2013/04/12 20:47:23.73 UTC MINOR: DEBUG #2001 Base RADIUS
"RADIUS: Transmit
Access-Request(1) 172.31.1.2:1812 id 70 len 206 vrid 1 pol authentication-2
USER NAME [1] 24 subscriber12@domain2.com
NAS IP ADDRESS [4] 4 192.0.2.1
SERVICE TYPE [6] 4 Framed(2)
FRAMED PROTOCOL [7] 4 PPP(1)
CHAP PASSWORD [3] 17 1 0xd4b73e0a17c0ad7f03c19bc1db5c291d
CHAP CHALLENGE [60] 41
0x620fa5f8be193d2066f6abad96c7de2df03986e3421f9733220d9520137b0bf40b30edc9c92bea30a2
VSA [26] 29 DSL(3561)
AGENT CIRCUIT ID [1] 13 circuit-id-12
AGENT REMOTE ID [2] 12 remote-id-12
NAS PORT ID [87] 11 3/2/2:1.100
CALLING STATION ID [31] 17 00:00:64:01:02:03
NAS IDENTIFIER [32] 5 BNG-1
NAS PORT TYPE [61] 4 PPPoEoQinQ(34)
```

If the subscriber can be authenticated and authorized, RADIUS responds with an Access-Accept containing attributes for both the control channel and the dynamic data service.

```
10 2013/04/12 20:47:23.73 UTC MINOR: DEBUG #2001 Base RADIUS
"RADIUS: Receive
  Access-Accept(2) id 70 len 211 from 172.31.1.2:1812 vrid 1 pol authentication-2
    VSA [26] 14 Alcatel(6527)
      SUBSC ID STR [11] 12 pppoe-user12
    FRAMED IP ADDRESS [8] 4 10.2.1.200
    VSA [26] 8 Alcatel(6527)
      DYN SERV SAP ID [164] 6 1 #:#.1
    VSA [26] 118 Alcatel(6527)
     DYN SERV SCRIPT PARAMS [165] 116 1 business_epipe={'t':('EPipe-CustomerName','Custom-
erName-Circuit-1','3','3','64496','192.0.2.5','192.0.2.1','3333')}
    VSA [26] 21 Alcatel(6527)
      DYN SERV POLICY [167] 19 1 dynamic-services-2
```

The existence of the Dyn Serv VSAs in the response triggers the BNG to start the execution of the Python script, but first the control channel session is completely established and an accounting start message is send to RADIUS. This is a standard accounting message for ESM subscribers.

```
11 2013/04/12 20:47:23.75 UTC MINOR: DEBUG #2001 Base RADIUS
"RADIUS: Transmit
  Accounting-Request(4) 172.31.1.2:1813 id 108 len 191 vrid 1 pol accounting-2
    STATUS TYPE [40] 4 Start(1)
    NAS IP ADDRESS [4] 4 192.0.2.1
    SERVICE TYPE [6] 4 Framed(2)
    FRAMED PROTOCOL [7] 4 PPP(1)
    FRAMED IP ADDRESS [8] 4 10.2.1.200
    FRAMED IP NETMASK [9] 4 255.255.255.255
    NAS IDENTIFIER [32] 5 BNG-1
    SESSION ID [44] 22 D6E559000000D2516872DB
    MULTI SESSION ID [50] 22 D6E559000000D3516872DB
```

```
    EVENT TIMESTAMP [55] 4 1365799643
    NAS PORT TYPE [61] 4 PPPoEoQinQ(34)
    NAS PORT ID [87] 11 3/2/2:1.100
    VSA [26] 29 DSL(3561)
      AGENT CIRCUIT ID [1] 13 circuit-id-12
      AGENT REMOTE ID [2] 12 remote-id-12
    VSA [26] 14 Alcatel(6527)
      SUBSC ID STR [11] 12 pppoe-user12
"
```

Next, the creation of the dynamic data service starts. As this is the first SAP for this service, the script which we reviewed above first creates the service instance.

```
12 2013/04/12 20:47:23.74 UTC MINOR: DEBUG #2001 Base dyn-script cli 1/1
"dyn-script cli 1/1: epipe:EPipe-CustomerName(cli 172 dict 0->31)

configure
  service
    epipe 1000 customer 1 create
      service-name "EPipe-CustomerName"
      description "EPipe-CustomerName"
      no shutdown
    exit
  exit
exit
"
```

Next, the SAP and the SDP are created within this service by the main function.

```
14 2013/04/12 20:47:23.74 UTC MINOR: DEBUG #2001 Base dyn-script cli 1/1
"dyn-script cli 1/1: business_epipe:3/2/2:1.1(cli 418 dict 0->308)

configure
  service
    epipe 1000
      sap 3/2/2:1.1 create
        description "CustomerName-Circuit-1"
        ingress
          qos 3
        exit
        egress
          qos 3
        exit
      exit
      spoke-sdp-fec 1000 fec 129 aii-type 2 create
        pw-template-bind 2
        saii-type2 64496:192.0.2.1:3333
        taii-type2 64496:192.0.2.5:3333
        no shutdown
      exit
    exit
  exit
exit
"
```

The service is created and is now active. As two RADIUS accounting destinations are configured in the dynamic services policy a RADIUS Accounting-Start message is sent to each destination to indicate the service is up.

```
16 2013/04/12 20:47:23.76 UTC MINOR: DEBUG #2001 Base RADIUS
"RADIUS: Transmit
  Accounting-Request(4) 172.31.1.2:1813 id 252 len 294 vrid 1 pol radius-server-policy-2
    STATUS TYPE [40] 4 Start(1)
    NAS IP ADDRESS [4] 4 192.0.2.1
    SESSION ID [44] 22 D6E559000000D4516872DB
    NAS PORT ID [87] 9 3/2/2:1.1
    DELAY TIME [41] 4 0
    NAS IDENTIFIER [32] 5 BNG-1
    EVENT TIMESTAMP [55] 4 1365799643
    MULTI SESSION ID [50] 22 D6E559000000D1516872DB
    USER NAME [1] 24 subscriber12@domain2.com
    VSA [26] 29 DSL(3561)
      AGENT CIRCUIT ID [1] 13 circuit-id-12
      AGENT REMOTE ID [2] 12 remote-id-12
    VSA [26] 117 Alcatel(6527)
      DYN SERV SCRIPT PARAMS [165] 115 business_epipe={'t':('EPipe-CustomerName','Custom-
erName-Circuit-1','3','3','64496','192.0.2.5','192.0.2.1','3333')}
"

15 2013/04/12 20:47:23.76 UTC MINOR: DEBUG #2001 Base RADIUS
"RADIUS: Transmit
  Accounting-Request(4) 172.31.1.2:1813 id 251 len 294 vrid 1 pol radius-server-policy-2
    STATUS TYPE [40] 4 Start(1)
    NAS IP ADDRESS [4] 4 192.0.2.1
    SESSION ID [44] 22 D6E559000000D4516872DB
    NAS PORT ID [87] 9 3/2/2:1.1
    DELAY TIME [41] 4 0
    NAS IDENTIFIER [32] 5 BNG-1
    EVENT TIMESTAMP [55] 4 1365799643
    MULTI SESSION ID [50] 22 D6E559000000D1516872DB
    USER NAME [1] 24 subscriber12@domain2.com
    VSA [26] 29 DSL(3561)
      AGENT CIRCUIT ID [1] 13 circuit-id-12
      AGENT REMOTE ID [2] 12 remote-id-12
    VSA [26] 117 Alcatel(6527)
      DYN SERV SCRIPT PARAMS [165] 115 business_epipe={'t':('EPipe-CustomerName','Custom-
erName-Circuit-1','3','3','64496','192.0.2.5','192.0.2.1','3333')}
"
```

For both RADIUS accounting destinations the interim accounting updates are also configured.

```
21 2013/04/12 20:51:46.69 UTC MINOR: DEBUG #2001 Base RADIUS
"RADIUS: Transmit
  Accounting-Request(4) 172.31.1.2:1813 id 173 len 511 vrid 1 pol radius-server-policy-1
    STATUS TYPE [40] 4 Interim-Update(3)
    NAS IP ADDRESS [4] 4 192.0.2.1
    SESSION ID [44] 22 D6E559000000D4516872DB
    NAS PORT ID [87] 9 3/2/2:1.1
    DELAY TIME [41] 4 0
    NAS IDENTIFIER [32] 5 BNG-1
    EVENT TIMESTAMP [55] 4 1365799906
    SESSION TIME [46] 4 125174
    MULTI SESSION ID [50] 22 D6E559000000D1516872DB
```

```
    USER NAME [1] 23 subscriber12@domain2.com
    VSA [26] 27 DSL(3561)
      AGENT CIRCUIT ID [1] 12 circuit-id-12
      AGENT REMOTE ID [2] 11 remote-id-12
    VSA [26] 241 Alcatel(6527)
      DYN SERV SCRIPT PARAMS [165] 115 business_epipe={'t':('EPipe-CustomerName','Custom-
erName-Circuit-1','3','3','64496','192.0.2.5','192.0.2.1','3333')}
      INPUT_INPROF_OCTETS_64 [19] 10 0x00010000000000000000
      INPUT_OUTPROF_OCTETS_64 [20] 10 0x00010000000000000000
      INPUT_INPROF_PACKETS_64 [23] 10 0x00010000000000000000
      INPUT_OUTPROF_PACKETS_64 [24] 10 0x00010000000000000000
      INPUT_HIGH_OCTETS_OFFER_64 [73] 10 0x00010000000000000000
      INPUT_LOW_PACK_OFFER_64 [76] 10 0x00010000000000000000
      INPUT_HIGH_PACK_OFFER_64 [75] 10 0x00010000000000000000
      INPUT_LOW_OCTETS_OFFER_64 [74] 10 0x00010000000000000000
      INPUT_UNC_PACK_OFFER_64 [78] 10 0x00010000000000000000
      INPUT_UNC_OCTETS_OFFER_64 [77] 10 0x00010000000000000000
      INPUT_HIGH_PACK_DROP_64 [71] 10 0x00010000000000000000
      INPUT_LOW_PACK_DROP_64 [72] 10 0x00010000000000000000
      INPUT_HIGH_OCTETS_DROP_64 [69] 10 0x00010000000000000000
      INPUT_LOW_OCTETS_DROP_64 [70] 10 0x00010000000000000000
      OUTPUT_INPROF_OCTETS_64 [21] 10 0x000100000000000033c
    VSA [26] 84 Alcatel(6527)
      OUTPUT_OUTPROF_OCTETS_64 [22] 10 0x00010000000000000000
      OUTPUT_INPROF_PACKETS_64 [25] 10 0x000100000000000000b
      OUTPUT_OUTPROF_PACKETS_64 [26] 10 0x00010000000000000000
      OUTPUT_INPROF_PACK_DROP_64 [81] 10 0x00010000000000000000
      OUTPUT_OUTPROF_PACK_DROP_64 [82] 10 0x00010000000000000000
      OUTPUT_INPROF_OCTS_DROP_64 [83] 10 0x00010000000000000000
      OUTPUT_OUTPROF_OCTS_DROP_64 [84] 10 0x00010000000000000000
"

19 2013/04/12 20:48:56.69 UTC MINOR: DEBUG #2001 Base RADIUS
"RADIUS: Transmit
  Accounting-Request(4) 172.31.1.2:1813 id 253 len 241 vrid 1 pol radius-server-policy-2
    STATUS TYPE [40] 4 Interim-Update(3)
    NAS IP ADDRESS [4] 4 192.0.2.1
    SESSION ID [44] 22 D6E559000000D4516872DB
    NAS PORT ID [87] 9 3/2/2:1.1
    DELAY TIME [41] 4 0
    NAS IDENTIFIER [32] 5 BNG-1
    EVENT TIMESTAMP [55] 4 1365799736
    SESSION TIME [46] 4 125004
    MULTI SESSION ID [50] 22 D6E559000000D1516872DB
    USER NAME [1] 23 subscriber12@domain2.com
    VSA [26] 27 DSL(3561)
      AGENT CIRCUIT ID [1] 12 circuit-id-12
      AGENT REMOTE ID [2] 11 remote-id-12
    VSA [26] 61 Alcatel(6527)
      DYN SERV SCRIPT PARAMS [165] 115 business_epipe={'t':('EPipe-CustomerName','Custom-
erName-Circuit-1','3','3','64496','192.0.2.5','192.0.2.1','3333')}
"
```

The “Stats-Type” in the dynamic service policy (or obtained via RADIUS in a VSA) defines what information is sent back to the accounting server (per server). In this example one was set to Stats-Type “time” and the other to “volume-time”. The first accounting message displays the content of “volume-time”. A full set of statistics counters per service class are provided for the dynamic

service. This is equivalent to the extended accounting statistics also provided in the ESM context. The second accounting message shows the content of “time”. No volume statistics counters are provided in this case.

Once the dynamic data services are instantiated they can be displayed with the regular show commands.

```
A:BNG-1# show service service-using
===============================================================================
Services
===============================================================================
ServiceId    Type      Adm  Opr  CustomerId Service Name
-------------------------------------------------------------------------------
1            VPLS      Up   Up   1          VPLS_For_Capture_SAPs
2            VPRN      Up   Up   1          VPRN_Control_Channel
3            VPRN      Up   Up   1          VPRN_REsidential_Subs
4            IES       Up   Up   1
10           VPRN      Up   Up   1
99           Mirror    Up   Up   1
500          Mirror    Up   Up   1
[1000]       Epipe     Up   Up   1          EPipe-CustomerName
[1001]       VPLS      Up   Up   1          VPLS-CustomerName
[1002]       IES       Up   Up   1          IES-CustomerName
[5000]       IES       Up   Up   1          IES-5000
[9999]       VPRN      Up   Up   1          VPRN-CustomerName
10001        VPLS      Up   Up   1
10002        Epipe     Up   Up   1
-snip-
-------------------------------------------------------------------------------
Matching Services : 20
-------------------------------------------------------------------------------
Dynamic Services : 5, indicated by [<svc-id>]
-------------------------------------------------------------------------------
===============================================================================
```

The dynamically created services are shown in the standard service list with their service IDs between brackets. It is possible to filter only the dynamic services using the **origin dyn-script** option.

```
A:BNG-1# show service service-using origin dyn-script
===============================================================================
Services
===============================================================================
ServiceId    Type      Adm  Opr  CustomerId Service Name
-------------------------------------------------------------------------------
[1000]       Epipe     Up   Up   1          EPipe-CustomerName
[1001]       VPLS      Up   Up   1          VPLS-CustomerName
[1002]       IES       Up   Up   1          IES-CustomerName
[5000]       IES       Up   Up   1          IES-5000
[9999]       VPRN      Up   Up   1          VPRN-CustomerName
-------------------------------------------------------------------------------
Matching Services : 5
-------------------------------------------------------------------------------
Dynamic Services : 5, indicated by [<svc-id>]
-------------------------------------------------------------------------------
```

```
===============================================================================
```

Similarly, the active SAPs can also be shown with the regular command.

```
A:BNG-1# show service sap-using
===============================================================================
Service Access Points
===============================================================================
PortId                          SvcId      Ing.  Ing.    Egr.  Egr.   Adm  Opr
                                           QoS   Fltr    QoS   Fltr
-------------------------------------------------------------------------------
3/2/1:*.100                     1          1     none    1     none   Up   Up
3/2/1:*.200                     1          1     none    1     none   Up   Up
3/2/2:*.100                     1          1     none    1     none   Up   Up
[3/2/1:4.100]                   2          1     none    1     none   Up   Up
[3/2/2:1.100]                   2          1     none    1     none   Up   Up
3/2/2:1000.1000                 2          1     none    1     none   Up   Up
[3/2/1:2.200]                   3          1     none    1     none   Up   Up
[3/2/1:3.200]                   3          1     none    1     none   Up   Up
3/2/1:1001.1001                 3          1     none    1     none   Up   Up
3/2/2:500.500                   3          1     none    1     none   Up   Up
3/2/2:100.100                   4          1     none    1     none   Up   Up
3/2/2:99.99                     99         1     none    1     none   Up   Up
[3/2/2:1.1]                     [1000]     3     none    3     none   Up   Up
[3/2/2:1.3]                     [1001]     3     none    3     none   Up   Up
[3/2/2:1.4]                     [1002]     5     ip4+ip6 6     ip4+i* Up   Up
[3/2/1:4.3]                     [5000]     1     none    1     none   Up   Up
[3/2/2:1.2]                     [9999]     3     ip4     3     ip4    Up   Up
3/2/1:99.99                     10001      1     none    1     none   Up   Up
3/2/19:100                      10001      1     none    1     none   Up   Up
3/2/20:100                      10001      1     none    1     none   Up   Up
-snip-
-------------------------------------------------------------------------------
Number of SAPs : 31
-------------------------------------------------------------------------------
Number of Managed SAPs : 4, indicated by [<sap-id>]
-------------------------------------------------------------------------------
Number of Dynamic Service SAPs : 5, indicated by [<sap-id>] [<svc-id>]
-------------------------------------------------------------------------------
===============================================================================
* indicates that the corresponding row element may have been truncated.
```

The description at the end of this show command explains how the dynamic services SAPs are displayed. Note that there are managed SAPs created for the control channel as well as dynamic data services SAPs.

If only the SAPs for dynamic data services should be displayed, the command **show service sap-using dyn-script** can be used.

```
A:BNG-1# show service sap-using dyn-script
===============================================================================
Service Access Points
===============================================================================
PortId                          SvcId      Ing.  Ing.    Egr.  Egr.   Adm  Opr
                                           QoS   Fltr    QoS   Fltr
```

```
-------------------------------------------------------------------------------
[3/2/2:1.1]                     [1000]     3     none    3     none   Up   Up
[3/2/2:1.3]                     [1001]     3     none    3     none   Up   Up
[3/2/2:1.4]                     [1002]     5     ip4+ip6 6     ip4+i* Up   Up
[3/2/1:4.3]                     [5000]     1     none    1     none   Up   Up
[3/2/2:1.2]                     [9999]     3     ip4     3     ip4    Up   Up
-------------------------------------------------------------------------------
Number of SAPs : 5
-------------------------------------------------------------------------------
Number of Dynamic Service SAPs : 5, indicated by [<sap-id>] [<svc-id>]
-------------------------------------------------------------------------------
===============================================================================
* indicates that the corresponding row element may have been truncated.
```

Conclusion

RADIUS-based dynamic data services provide an innovative way for business service provisioning. They are created both automatically and instantaneously.

It removes the need for comprehensive integration tasks into the existing IT environment for service provisioning and therefore speeds up the introduction of new service offerings, and thus new revenue streams for the customer.