3 Function—VPN lifecycle

3.1 Definitions

Definition 1 **Fixed VPN**

A Fixed_VPN is defined to comprises a *Membership_Specification* **and** *a Comon_QoS* **and** a set of *ToS_QoS*. A Common_QoS is a *QoS_Spec*. A ToS_QoS is a *{ToS, QoS_Spec}* tuple, **and** where no ToS_QoS for the VPN specifies the same *ToS*.

Definition 2 **Membership_Specification**

A Membership_Specificiation is defined to comprise all of the following:

56 – An administrative state, being an one of the values {LOCKED, UNLOCKED}.

57 – A VPN_installation_state, being one value of the enumeration *{DESIGN, RESERVE, INSTALL}*[1].

58 – The *address_allocation_range* being an *address_range.*

59 – The set of UNIs, where each UNI is a physical port either on the customer side of the telco's edge router (*UNI-T*), or on the telco side of the customer's NTU (*UNI-C*).

60 – For each UNI, a UNI_installation_state, being one value of the enumeration *{DESIGN, RESERVE, INSTALL}*[2].

61 – For each UNI-C, the *bandwidth* in each direction of the CAN_link, where bandwidth is the peak rate measured in *bits* per second, where bits are payload bits for the link protocol.

62 – For each UNI, a *user_address*, being a *set of masked address range*, **and** where no address is a member of more than one user address for any UNI.

Definition 3 **CAN_link**

A CAN_link is defined to be a physical transmission bearer between the customer's premises and the telco premises.

Definition 4 **Address_Range**

An address_range is defined to be an set of contiguous IP addresses.

Definition 5 **Masked_Address_Range**

A masked_ significant address_range is defined to be a set of continuous IP

1. The implementation may also have other installation states, however these are the states that may be specified by the client.
2. The implementation may also have other installation states, however these are the states that may be specified by the client.

addresses, such that some defined number of most significant bits of each address are the same.

Definition 6

ToS

A ToS is defined to be the name of a IP type of service.

Definition 7

QoS_Spec

A QoS_Spec is defined to be:

63 - an optional *Q0_spec*, where a Q0_spec is a {*user_bandwidth, user_delay*} tuple.

64 - a set of Q1_specs, where each Q1_spec is a *{UNI, direction, user_bandwidth}* tuple **and** direction is either {INGRESS or EGRESS} **and** no Q1_spec in the set specifies the same *{UNI, direction}* tuple.

65 - a set of Q2_specs, where each Q2_spec is a {UNI, UNI, *directed_user_bandwidth}* tuple **and** directed_user_bandwidth is the user_bandwidth from the first UNI to the second **and** no Q2_spec in the set specifies the same pair of UNIs.

Definition 8

User_bandwidth

User_bandwidth is defined to be either a *constant_bitrate* or *burst_bitrate*. A constant_bitrate is defined to be a *user_bitrate*. A burst_bitrate is defined to be a *{base_bitrate, peak_bitrate, burst_occupancy, GOS}* tuple. Base_bitrate and peak_bitrate are defined to be a *user_bitrate*. Burst_occupancy is defined to be number in the range 0 to 1.0. GOS is defined to be a number in the range 0 to 1.0.

Definition 9

User_bitrate

User_bitrate is defined to be the number of IP header and payload bits generated by the user per unit time.

Definition 10

User_delay

User_delay is defined to be the delay from an IP packet being transmitted by the originating NTU to that packet being received by the destination NTU, when averaged over 100 packets being transmitted at the maximum rate allowed by the VPN specification.

Definition 11

Trail

A trail is defined to be the set of path objects required to implement a VPN. Router trail components, are the trail components that configure edge routers. Non-router trail components are any trail component that is not a router trail component.

3.2 Feature—VPN life-cycle operations

3.2.1 Creating, deleting, changing an optimizing a VPN

66

Specifying a VPN specification

Requirement 1 CoreIP shall allow a client to create an empty VPN in DESIGN state. CoreIP shall reject the creation of a VPN that is not empty or one that is not in DESIGN state.

Requirement 2 CoreIP shall allow a client to specify each of the following independent changes to a VPN, as one or more Gnm operations:

Requirement 2.1 · change the address_allocation_range

Requirement 2.2 · add a UNI specifying the user_address

Requirement 2.3 · remove a UNI

Requirement 2.4 · specify the CAN_link bandwidth of a UNI-C

Requirement 2.5 · change the common_QoS by—changing the Q0_spec, adding or removing a Q1_spec or adding or removing a Q2_spec.

Requirement 2.6 · add or remove a Tos_QoS

Requirement 2.7 · change a ToS_QoS, by—changing the Q0_spec, adding or removing a Q1_spec or adding or removing a Q2_spec.

67

The transactional model for changes

Requirement 3 CoreIP shall allow a client to change an existing VPN in either a *transactional* or *non-transactional* model. The non-transactional model specifies an operation that

is expressed in exactly one Gnm operation, and CoreIP *actions* the change when the operation is requested. The transactional model executes as follows:

Requirement 3.1 · A client starts a transaction on a pre-existing VPN.

Requirement 3.2 · CoreIP opens the transaction.

Requirement 3.3 · A client issues change requests[3], and CoreIP takes no action and raises no errors.

Requirement 3.4 · The client commits the transaction. CoreIP then attempts to *action* the accumulated result of the changes expressed by the client.

Requirement 3.5 · If CoreIP successfully actions the change, then CoreIP shall indicate to the client that the change is successful **and** close the transaction.

Requirement 3.6 · If CoreIP does not successfully action the change, then CoreIP indicates to the client that the change is unsuccessful[4].

Requirement 4 CoreIP shall at any time allow a client to cancel an action in progress or a transaction that is open. If there is an action in progress, CoreIP shall *rollback* the action. If there is a transaction that is open, CoreIP shall close the transaction **and** shall revert the state of the VPN to that as at the time the transaction was started. CoreIP shall raise an error to the client that initiated the action[5].

68

Actioning a change

Requirement 5 While actioning a change on a particular VPN, CoreIP shall reject any requests to change or delete, test or repair the VPN.

Requirement 6 CoreIP shall action a change, where the new VPN_installation_state is DESIGN, by deleting any components of the trail.

Requirement 7 CoreIP shall action a change, where the new VPN_installation state is INSTALL or RESERVE, by performing the following step in the following logical order[6]:

Requirement 7.1 · If any UNI has UNI_installation_state DESIGN, then ignore that UNI and any Q1 or Q2 specification regarding that UNI from the action.

Requirement 7.2 · If the client specification of the VPN does not conform to the specification of a fixed_VPN, then raise an error and fail the action.

Requirement 7.3 · If the specification of the VPN is *inconsistent*, then raise an error and fail the operation.

Requirement 7.4 · *Design* the trail. If the VPN installation state is RESERVED, each trail component shall have installation state RESERVED. If the

3. Any changes made by any client are part of the transaction. There is no concept of a transaction being exclusive to a particular client.

4. And does not close or roll-back the transaction.

5. That is, the client that committed the transaction, or the non-transactional client that executed the operation.

6. That is, any algorithm that produces the same client-visible outputs as specified here, in particular the hierarchical uncovering of error conditions.

VPN_installation_state is INSTALLED, then the trail components that solely support traffic to a UNI that has RESERVED UNI_installation_state, or solely support traffic between UNIs that have RESERVED UNI_installation_state, shall have installation state RESERVED, and other trail components shall have installation state INSTALLED. Each trail component shall have the same administrative state as the VPN.

Requirement 7.5 · If the design cannot be done, then raise an error and fail the operation.

Requirement 7.6 · Execute the *minimal network change to* implement the new VPN. If this succeeds, then the action has succeeded. If this fails, then *rollback* the change.

Requirement 8 CoreIP shall regard a VPN specification as *inconsistent*, if any of the following are true:

Requirement 8.1 · The VPN specifies a UNI that is part of another VPN, or otherwise not free on the router.

Requirement 8.2 · A specified UNI does not exist.

Requirement 8.3 · A UNI, or a UNI's edge router is preconfigured, and the new VPN_installation_state is INSTALLED and the new UNI_installation_state is INSTALLED.

Requirement 8.4 · A CAN_link does not have sufficient capacity to support the specification.

Requirement 8.5 · The Common_QoS, or the QoS for any ToS, is incomplete, as defied by Reference 1.

Requirement 9 CoreIP shall rollback a change by the following process:

Requirement 9.1 · No further operations, other than rollback operations, shall be commenced.

Requirement 9.2 · The VPN is regarded as *modification disrupted*[7].

Requirement 9.3 · Any operations that are in progress shall be cancelled.

Requirement 9.4 · Any operations that have been completed, or complete in spite or cancellation, shall be undone by deleting any created trail components, re-creating any deleted trail components and changing back any changed trail components.

Requirement 9.5 · If any undo operation fails, then CoreIP shall raise an error to the client that initiated the action[8] and shall assert and alarm on the VPN. The alarm shall indicate that the VPN has reduced function and the trail component that is in question. The alarm shall remain asserted until a successful change or optimize operation is executed on the VPN.

Requirement 9.6 · If all undo operations succeed, or if any fail, then the VPN ceases to be *modification disrupted*. If any undo operation fails, then the VPN becomes *corrupted*.

7. See 'Tracking VPN state' on page 16

8. If the rollback is a result of a cancel, then the error does not got to the client that did the cancel, but the client that committed the transaction, or executed the non-transactional operation.

Requirement 10 If there are no trail components existing for the VPN, then CoreIP shall execute a minimal network change by creating the new trail components.

Requirement 11 If there are existing trail components for the VPN, then CoreIP shall execute a minimal network change by executing the following steps in the following logical sequence:

Requirement 11.1 · Any components in the new trail, that also exist with the same state in the old trail are ignored.

Requirement 11.2 · Any non-router components that exist in the new trail, **and** for which there is no component in the old trail with the same terminations, shall be created. Any non-router components that exist in the new trail, **and** for which there is a component in the old trail with the same terminations **and** the new component has higher capacity, shall be changed nondisruptively to the new state.

Requirement 11.3 · If there are any non-disruptive changes that could not be executed and raised InvalidImpact exceptions, will be executed, this time as disruptive changes. If this step is executed, then the VPN is *modification disrupted* prior to executing the changes.

Requirement 11.4 · Any router components that exist in the new trail, but not in the old, shall be created. The VPN is *modification disrupted*[9] at the start of this step.

Requirement 11.5 · Any router components that exist in the old trail, but not in the new, shall be deleted. Any non-router components that exist in the new trail, **and** for which there is a component in the old trail with the same terminations **and** the new component has lower capacity, shall be changed nondisruptively to the new state. If the nondisruptive change fails, then attempt the same change disruptively. The VPN ceases to be *modification disrupted* at the successful completion of this step.

Requirement 11.6 · Any non-router components that exist in the old trail, **and** for which there is no component in the new trail with the same terminations, shall be deleted. Any non-router components that exist in the old trail, **and** for which there is a component in the new trail with the same terminations **and** the new component does not have higher capacity, shall be changed to the new state.

69

Deleting a VPN

Requirement 12 CoreIP shall allow a client to delete a specified VPN. If the VPN is not in DESIGN state, then CoreIP shall *recover*. If recovery is attempted successfully, or is not attempted, then CoreIP shall change the VPN_installation_state to DELETED.

Requirement 13 CoreIP shall allow a client to view the state of a DELETED VPN, but for all other purposes the a DELETED VPN is regarded as non-existent.

9. See ‘Tracking VPN state’ on page 16

Requirement 14 CoreIP shall recover, by deleting all trail components. If this succeeds, then the recover has succeeded. If this fails, then *rollback* the recovery.

Requirement 15 While recovering a particular VPN, CoreIP shall reject any requests to change, delete test or repair the VPN.

Requirement 16 If a VPN has any pending alarms at the time it is successfully deleted, then it shall issue a alarm clearing to de-assert any currently asserted alarms.

70

Optimizing a VPN

Requirement 17 CoreIP shall allow a client to optimize a VPN. CoreIP shall execute this as a null change action, with the exception that the design process is obliged to take no account of the current trail of the VPN[10].

71

Preview a change

Requirement 18 CoreIP shall allow a client to preview a change operation. This may be done as a non-transactional operation, or as a transactional operation, by previewing the commit operation.

Requirement 19 CoreIP shall preview a change, where the new VPN_installation_state is DESIGN, by always accepting the operation.

Requirement 20 CoreIP shall preview a change, where the new VPN_installation state is INSTALL or RESERVE, by performing the same operations as specified for actioning a change, with the exception that the *minimal network change*, is replaced by the following steps:

Requirement 20.1
- Ensure that each edge router can be contacted, if not then fail the preview.

Requirement 20.2
- For each non-router trail that exist in the new trail, **and** for which there is no component in the old trail with the same terminations, then preview the creation of the component. If that fails, then fail the preview.

Requirement 20.3
- For each non-router components that exist in the new trail, **and** for which there is a component in the old trail with the same terminations then preview the change as a disruptive change.

10. The design of a non-optimized change could use the existing trail to bias the choice of the new trail.

3.2.2 Tracking VPN state

72

Reading state

Requirement 21

CoreIP shall allow a client to obtain the state of a VPN. This state shall include all the following:

Requirement 21.1

- The VPN specification, cost and trail applicable at the time of enquiry. If a transaction is in progress, then the applicable state is that prior to the transaction starting, else if an action is in progress, then the applicable state is that prior to starting the action, else the applicable state is the current state.

Requirement 21.2

- Whether a transaction is open on the VPN.

Requirement 21.3

- Whether an action or recovery is in progress, and if so the internal or delegated activities that are currently started but not finished.

73

State alarms

Requirement 22

For the duration that a VPN is *modification disrupted*, CoreIP shall assert a *modification disruption* alarm on the VPN, indicating mild dysfunction and stating the cause.

Requirement 23

For the duration that a VPN is *corrupted*, CoreIP shall assert a *corruption* alarm on the VPN, indicating total dysfunction and stating the cause.

Requirement 24

CoreIP shall support a *long_transaction_alarm* configuration rule which specifies a delay in seconds, and the content of an alarm. The rule may be qualified to the fixed_VPN network only[11]. While a transaction is open for longer than the period specified in the rule, then CoreIP shall assert a *long transaction* alarm on the VPN indicating mildly dysfunction and stating the cause.

74

Notifications

Requirement 25

CoreIP shall generate the notifications described in Table 1 (p. 17). The notification shall emanate from the object specified in *generated by*, the

11. Though later releases may relax this.

notification shall be generated at the time specified in *generated when*, and it shall have content as specified in *content*.

Table 1. **VPN state notifications**

Notification	Event	Generated		Content
		by	when	
Lifecycle	A VPN is created or deleted.	Containing network	completion of successful transaction or operation	The VPN and the event.
State change	A VPN has changed any of its specification.	VPN		The VPN, and the type of state that has changed, as one or more of: • adding or removing UNIs, nominating the UNIs • changing the Q0 • changing the Q1, nominating the UNI • changing Q2, nominating both UNIs • changing installation or administrative state.
Transaction	A transaction starts, commits, commit fails, completes or is cancelled or, an operation starts, completes or is cancelled.	VPN	when event occurs	The VPN and the event and if it is a failure the error raised to the invoking client.

3.3 Feature—Capability enquiry

3.3.1 Feature exposure

Requirement 26 CoreIP shall present features as specified by the Gnm model

Requirement 27 CoreIP shall present a feature interaction model, as specified by the Gnm model.

3.3.2 UNI enquiry and allocation

Requirement 28 CoreIP shall allow a client to view of all UNIs, and their state.

Requirement 29 CoreIP shall allow a client to request the identity of the best available UNI on a router or router group[12]. The best UNI is the *equi-cost choice* from all UNI meeting all the following candidate criteria:

Requirement 29.1 · The UNI shall be a member of the sub-tree specified by the client.

Requirement 29.2 · The UNI shall a either a UNI-C or UNI-T, as specified by the client.

Requirement 29.3 · The UNI shall only be preconfigured, if there is no non-preconfigured candidate.

Requirement 29.4 · The UNI shall not be a member of any VPN, nor shall it be busy for any other purpose.

Requirement 30 CoreIP shall support a *termination allocation* configuration rule[13], specifying a {*role, cost_differential, allocation_policy*} tuple. Role may have the values UNI or ANY. This rule may be qualified by a router or router group.

Requirement 31 CoreIP shall implement equi-cost choice as follows:

Requirement 31.1 · If there is no active termination allocation rule, then return one of the UNIs with the lowest cost.

Requirement 31.2 · If there is a active termination allocation rule, then create the *equi-cost set* to be the subset of candidates, who cost no more than *(1+cost differential)* times the cost of the lowest cost termination.

Requirement 31.3 · If the allocation_policy is DONT_CARE, then return any member of the equi-cost set.

Requirement 31.4 · If the allocation_policy is RANDOM, then return a randomly selected member of the equi-cost set.

Requirement 31.5 · If the allocation policy is EMPTY_COUNT_BIAS, then return a member of the equi-cost set which was returned by a subnetwork containing the fewest occupied UNIs.

Requirement 31.6 · If the allocation policy is FULL_COUNT_BIAS, then return a member of the equi-cost set which was returned by a subnetwork containing the most occupied UNIs.

12. See ‘Feature—Router life cycle’ on page 22.

13. This is specified to be compatible with CoreConnect.

3.3.3 Address allocation

Requirement 32 CoreIP shall allow a client to request a free IP address to assign to a UNI. CoreIP shall return an address that is a member of the VPN's address_allocation_range, and is not currently assigned to any UNI in the VPN.

3.3.4 Object search

Requirement 33 CoreIP shall allow a client to determine the VPNs that use a specified UNI, CAN-link, router, non-router trail component or recursive sub-component of a non-router trail component.

4 Function—Service assurance

4.1 Feature—VPN alarm and operational state

Requirement 34 CoreIP shall maintain an *alarm state* for each VPN. When the alarm state changes, CoreIP will issue an alarm containing the information from the alarm state.

Requirement 35 CoreIP shall support a *alarm aggregation* plug-in. This plug-in shall take as input, the VPN and the set of *{alarm_state, trail component}* tuples for each trail component, and shall output the alarm state severity, causal alarm, probable cause, specific problem and operational state[14].

Requirement 36 If there is no alarm aggregation plug-in, CoreIP shall generate the alarm state as follows:

Requirement 36.1 · *severity,* as the highest severity of all trail component active alarms

Requirement 36.2 · if there is an active alarm, then *causal alarm*, as one of the active alarms with the highest severity.

Requirement 36.3 · if there is a active alarm, then probable cause, specific problem and additional text fields will be the respective values from the dysfunctional alarm rule.

Requirement 36.4 · the *operational state* is the severity.

Requirement 37 CoreIP shall allow a client to determine the operational state of the VPN.

14. Note that the intent is that CoreIP makes available to the plug-in all the subordinate alarm states, thus allowing all the plug-in to be stateless

4.2 Feature—Test and repair

4.2.1 General test and repair

Requirement 38 While actioning a test or repair on a particular VPN, CoreIP shall reject any requests to change or delete, test or repair the VPN.

Requirement 39 CoreIP shall publish its internal and plug-in- test and repair functions, in the manner defined by the Gnm interface.

Requirement 40 CoreIP shall support a plug-in that allows a client to provide the implementation of a Gnm test or repair operation, and the signature of the operation for publication via the Gnm interface.

4.2.2 Internal tests

75 **Referential test**

Requirement 41 CoreIP shall implement the *referential internal test* by performing the following sub-tests. If any sub-test fails, then the referential test has failed and the client will be informed of the failure and the reason. The sub-tests are:

Requirement 41.1 · Verify that each trail component exists.

Requirement 41.2 · Verify that the state of each trail component is consistent with that set when it was created. For routers trail components, verification must read the state from the router.

Requirement 41.3 · Verify that no additional trails exist on UNIs and any internal terminations that comprise the VPN.

Requirement 41.4 · For each non-router component, delegate the referential test on that component.

Requirement 42 CoreIP shall regard a VPNs that fails a referential integrity test as *corrupted*, and VPN that passes the referential integrity test as not corrupted.

76 **Name**

Requirement 43 CoreIP shall implement the *name internal test* by ensuring that the external name service retains the state as set by CoreIP. If it does not, then inform the user of the names that are missing, and the names that refer to the wrong object. If there are a large number of these, then limit the number of reported inconsistencies to the number requested by the client.

4.2.3 Internal repairs

77

Reconfigure routers

Requirement 44

CoreIP shall implement the *router reconfiguration repair*, where the client specifies at least one of the following filters, a particular router, a particular VPN, a particular UNI. The target UNIs, are intersection of UNIs corresponding to the specified filters.

Requirement 45

CoreIP shall execute the repair by removing from any router containing a target UNI any configuration referencing that UNI, and then rebuilding the configuration expected by CoreIP. If any steps in this process fail, then the CoreIP shall inform the client of the failure, and what has failed, otherwise the client shall be informed of success.

Requirement 46

CoreIP shall regard all VPNs that have a target UNI as *modification disrupted*[15] for the duration of the repair. If the repair fails, the VPN is *corrupted*.

78

Reconfigure inter-router

Requirement 47

CoreIP shall implement the *inter-router reconfiguration repair*, where the client specifies a particular VPN.

Requirement 48

CoreIP shall execute the repair by deleting any non-router trail that fails the referential integrity test, and then creating any non-router trails that are required by CoreIP. If any steps in this process fail, then the CoreIP shall inform the client of the failure and what has failed, otherwise it shall inform the client of success.

Requirement 49

CoreIP shall regard the VPN as *modification disrupted*[16] for the duration the repair. If the repair fails, the VPN is *corrupted*.

79

Reconfigure names

Requirement 50

CoreIP shall implement the *name server repair* by deleting any name server entries that point to the wrong object, and creating any names that are missing. If any of these steps fail, then CoreIP shall inform the client of failure and what has failed, otherwise it shall inform the client of success.

15. See ‘Tracking VPN state’ on page 16

16. See ‘Tracking VPN state’ on page 16

80 **Reset**

Requirement 51 CoreIP shall implement the *reset repair* by clearing any *transient state* that it holds about the routers, non-router trails, plug-ins or the database. The transient state is any state held by CoreIP other than that held in the database.

5 Function—Fabric life cycle

5.1 Feature—Router life cycle

81 **Router group lifecycle**

Requirement 52 CoreIP shall allow a client to define a *router group*. This will specify all of the following:

Requirement 52.1 · The containing router group, which may be either the *root router group* defined by CoreIP, or a client defined router group.

82 · The name of the router group. CoreIP shall reject the operation if this name is not globally unique.

Requirement 53 CoreIP shall allow a client to delete a router group that is empty, and that is not the root router group.

83 **Edge router lifecycle**

Requirement 54 CoreIP shall allow a client to add a router. This will specify all of the following:

Requirement 54.1 · The containing router group.

Requirement 54.2 · The IP address of the router's MIB.

Requirement 54.3 · Optionally, the value expected for SysObjectId of the management agent.

Requirement 55 CoreIP shall allow a client to delete a router that supports no VPNs.

Requirement 56 When a router is added, CoreIP shall attempt to contact the management agent. If that fails, then the router is preconfigured, otherwise it is not preconfigured. If a router is preconfigured, CoreIP shall continuously attempt to contact the management agent, by listening for traps and by period polling. CoreIP shall listen for traps if and only if the preconfiguration trap reception configuration rule has value *true*. The periodic polling period is set by the *preconfiguration polling period* configuration rule. Once the management agent has been contacted, then the router is no longer preconfigured.

Requirement 57 CoreIP shall support a *router preconfiguration complete* plug-in. This is called whenever the state of a router changes from preconfigured to configured. This plug-in is passed the identity of the router, and all VPNs that are currently using UNIs on that router.

Requirement 58 As soon as a management agent in a router has been contacted, CoreIP shall read the SysObjectId to determine the type of router. CoreIP shall continuosly attempt to read and update the type of router by period polling. The periodic polling period is set by the *router type polling period* configuration rule.

Requirement 59 CoreIP shall support the *router discovery rule*, specifying an set of *address_ranges*, a set of *object_ids*, a *poll period* and a *router locator plug-in*. CoreIP shall poll each address in the set of *address_ranges* with the period given in *poll_period*, for a SysObjectId. If a polled device has a SysObjectId with a value that is a member of *object_ids*, and if the router locator is specified, then CoreConnect shall call the specified router locator plug-in, passing the IP address, sysObjectId and sysLocation. If the plug-in returns the name of a router group, then CoreIP will create a router, in that router group.

Requirement 60 CoreIp shall support a *router-installation* plug-in[17]. This plug-in is called whenever a router is added (by a client, preconfigured or not, or by discovery) and the router type has been confirmed. The plug-in is passed the IP address of the router, the SysObjectId, the sysLocation.

Requirement 61 CoreIp shall support a router-removal plug-in. This plug-in is called whenever a router is removed. The plug-in is passed the IP address of the router, the SysObjectId, the sysLocation and any CAN_links that it currently has.

5.1.1 Feature—UNI lifecycle

Requirement 62 CoreIP shall allow a client to add a UNI. This will specify all of the following:

Requirement 62.1 · The containing router.

Requirement 62.2 · The MIB address of the UNI.

Requirement 63 CoreIP shall allow a client to delete a UNI that supports no VPNs, and for which the router is preconfigured, or the UNI is preconfigured, or the UNI is not physically installed. If these conditions do not hold, then CoreIP shall reject the deletion.

Requirement 64 When a UNI is added to a preconfigured router, CoreIP shall consider the UNI to be preconfigured. When a UNI is added to a non-preconfigured router, CoreIP shall attempt to contact the management agent and verify the existence of the hardware. If the hardware does not exist, then the UNI is preconfigured, otherwise it is not. If a UNI is preconfigured, and it is subject to an action (see ‘Creating,

17. This is typically used for link discovery.

deleting, changing an optimizing a VPN' on page 11), or to any other operation that enquires of the UNI state, then prior to basing decisions on whether the UNI is preconfigured or not, CoreIP shall attempt to contact the management agent and verify the existence of the hardware to determine if the UNI is still preconfigured.

Requirement 65 Whenever CoreIP executes an operation that returns a UNI[18], then CoreIP shall ensure that it considers UNIs that are preconfigured, UNIs that are part of VPNs, and UNIs that have installed hardware on non-preconfigured routers.

Requirement 66 CoreIP shall support a *UNI preconfiguration complete* plug-in. This is called whenever the state of a UNI changes from preconfigured to not preconfigured. This plug-in is passed the identity of the UNI, and all VPNs that are currently using the UNI.

Requirement 67 CoreIp shall support a *UNI-installation* plug-in. This plug-in is called whenever a UNI is added (by a client, preconfigured or not, or by discovery). The plug-in is passed the address of the router, and the UNI.

Requirement 68 CoreIp shall support a *UNI-removal* plug-in. This plug-in is called whenever a router is removed. The plug-in is passed the IP address of the UNI.

5.1.2 Feature—Link lifecycle

Requirement 69 CoreIP shall allow a client to add a UNI. This will specify all of the following:

Requirement 69.1 · The containing router.

Requirement 69.2 · The MIB address of the UNI.

Requirement 70 CoreIP shall allow a client to delete a UNI that supports no VPNs, and for which the router is preconfigured, or the UNI is preconfigured, or the UNI is not physically installed. If these conditions do not hold, then CoreIP shall reject the deletion.

6 TODO

84 · *design trail*

Requirement 70.1 · check Gnm for unspecified operations

Requirement 71 packetizing overheads

Requirement 72 equipment terms

Requirement 73 burst occupancy

18. Such as find_free_termination, get termination_by_name, get_termination_by index or get_contained_terminations.

Requirement 74 think about shared IRN

Requirement 75 mark planned-only functions

Requirement 76 magic Ip addrsses

Requirement 77