TECHNICAL REPORT

# TR-181
# Device Data Model

Issue: 2 Amendment 13
Issue Date: September 2019

## Notice

The Broadband Forum is a non-profit corporation organized to create guidelines for broadband network system development and deployment. This Technical Report has been approved by members of the Forum. This Technical Report is subject to change. This Technical Report is copyrighted by the Broadband Forum, and all rights are reserved. Portions of this Technical Report may be copyrighted by Broadband Forum members.

## Intellectual Property

Recipients of this Technical Report are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of this Technical Report, or use of any software code normatively referenced in this Technical Report, and to provide supporting documentation.

## Terms of Use

### 1. License

Broadband Forum hereby grants you the right, without charge, on a perpetual, non-exclusive and worldwide basis, to utilize the Technical Report for the purpose of developing, making, having made, using, marketing, importing, offering to sell or license, and selling or licensing, and to otherwise distribute, products complying with the Technical Report, in all cases subject to the conditions set forth in this notice and any relevant patent and other intellectual property rights of third parties (which may include members of Broadband Forum). This license grant does not include the right to sublicense, modify or create derivative works based upon the Technical Report except to the extent this Technical Report includes text implementable in computer code, in which case your right under this License to create and modify derivative works is limited to modifying and creating derivative works of such code. For the avoidance of doubt, except as qualified by the preceding sentence, products implementing this Technical Report are not deemed to be derivative works of the Technical Report.

### 2. NO WARRANTIES

THIS TECHNICAL REPORT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NONINFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS TECHNICAL REPORT SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE BROADBAND FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS TECHNICAL REPORT.

### 3. THIRD PARTY RIGHTS

Without limiting the generality of Section 2 above, BROADBAND FORUM ASSUMES NO RESPONSIBILITY TO COMPILE, CONFIRM, UPDATE OR MAKE PUBLIC ANY THIRD PARTY ASSERTIONS OF PATENT OR OTHER INTELLECTUAL PROPERTY RIGHTS THAT MIGHT NOW OR IN THE FUTURE BE INFRINGED BY AN IMPLEMENTATION OF THE TECHNICAL REPORT IN ITS CURRENT, OR IN ANY FUTURE FORM. IF ANY

SUCH RIGHTS ARE DESCRIBED ON THE TECHNICAL REPORT, BROADBAND FORUM TAKES NO POSITION AS TO THE VALIDITY OR INVALIDITY OF SUCH ASSERTIONS, OR THAT ALL SUCH ASSERTIONS THAT HAVE OR MAY BE MADE ARE SO LISTED.

The text of this notice must be included in all copies of this Technical Report.

## TR Issue History

| Issue Number | Approval Date | Publication Date | Issue Editor | Changes |
|---|---|---|---|---|
| Issue 2 | May 2010 | | Paul Sigurdson, Broadband Forum<br>William Lupton, 2Wire | Original. Defines version 2.0 of the TR-069 Device data model (Device:2.0). |
| Issue 2 Amendment 1 | November 2010 | | Paul Sigurdson, Broadband Forum<br>William Lupton, 2Wire | Added support for Software Module Management in the data model (no change to this document). Defines version 2.1 of the TR-069 Device data model (Device:2.1). |
| Issue 2 Amendment 2 | February 2011 | | Paul Sigurdson, Broadband Forum<br>William Lupton, Pace | Added support for IPv6 and Firewall in the data model (added IPv6 and Firewall Appendices to this document). Defines version 2.2 of the TR-069 Device data model (Device:2.2). |
| Issue 2 Amendment 3 | July 2011 | | | ***This update to TR-181 did not update this document; only the XML data model was updated.***<br><br>Added support for proxy management and alias-based addressing. |
| Issue 2 Amendment 4 | November 2011 | | William Lupton, Pace | ***This update to TR-181 did not update this document; only the XML data model was updated.***<br><br>Added support for G.hn and Optical interfaces in the data model, and additional WiFi parameters (updated interface stack figures). Defines version 2.4 of the TR-069 Device data model (Device:2.4). |
| Issue 2 Amendment 5 | May 2012 | | William Lupton, Pace | Added support for IPsec and bulk data collection in the data model (added Tunneling Annex and IPsec Appendix to this document). Defines version 2.5 of the TR-069 Device data model (Device:2.5). |

| Issue Number | Approval Date | Publication Date | Issue Editor | Changes |
|---|---|---|---|---|
| Issue 2 Amendment 6 | November 2012 | 18 January 2013 | Tim Carey, Alcatel-Lucent | Added support M2M SCL Administration as an Appendix.<br>Defines version 2.6 of the TR-069 Device data model (Device: 2.6). |
| Issue 2 Amendment 7 | 11 November 2013 | 17 January 2014 | Apostolos Papageorgiou, NEC<br>William Lupton, Cisco | Added ZigBee and Provider Bridge data models (including theory of operation); also added additional WiFi statistics, and other minor changes; added backup/restore theory of operation.<br>Defines version 2.7 of the TR-069 Device data model (Device: 2.7). |
| Issue 2 Amendment 8 | 8 September 2014 | 15 September 2014 | William Lupton, Cisco | Added LLDP and HTIP home network topology discovery parameters, G.997.1-2012 DSL parameters, various WiFi parameters (associated device statistics, retry limits, reports, QoS), IPv6-related IP diagnostics parameters, and other minor changes; updated G.hn data model to align with G.9962; updated Annex B on tunneling, and added GRE and MAP data models (including theory of operation); added PCP data model (including theory of operation); added Cellular interface data model.<br>Defines version 2.8 of the TR-069 Device data model (Device: 2.8). |
| Issue 2 Amendment 9 | 1 December 2014 | 11 February 2015 | Douglas Knisely, Qualcomm, Inc | ***This update to TR-181 did not update this document; only the XML data model was updated.***<br><br>Added support forWiFi MAC Address Filtering, fixes for Traceroute, IEEE 1905 data model and incorporated new components from TR-143 Amendment 1 |

| Issue Number | Approval Date | Publication Date | Issue Editor | Changes |
|---|---|---|---|---|
| Issue 2 Amendment 10 | 9 November 2015 | 13November 2015 | Klaus Wich, Axiros | Added data model updates:<br>- MQTT model<br>- Bulk data over HTTP<br>- DNS Server updates<br>- new diagnostics state |
| Issue 2 Amendment 11 | 18 July 2016 | 12 August 2016 | Klaus Wich, Axiros<br>Mark Tabry, Google | Added G.fast data model (including theory of operation).<br><br>Data model additions:<br>- LED status model<br>- Layer 2 tunnel support for IP diagnostics model<br>- DSL G.fast model<br>- Management Frame Protection support for WiFi model<br>- WPS 2.0 support for WiFi model<br>- User interface toggle<br>- User interface messaging model<br>- ConnectionRequest HTTP service toggle<br>- DNS fallback support for XMPP connections |
| Issue 2 Amendment 12 | 16 March 2018 | 9 May 2018 | Steve Nicolai, Arris | Added Appendix I, II, IV from TR-157a10 as Appendix XVII, XVIII and XIX.<br>Added Appendix XX BASAPM and LMAP Theory of Operations<br>Added Annex H from TR-069a5 as Annex C.<br><br>Data model additions:<br>- Ethernet Link Aggregation |
| Issue 2 Amendment 13 | 13 September 2019 | 13 September 2019 | Klaus Wich, Huawei | Unified text for CWMP and USP support, updated references.<br><br>Data model additions:<br>IoT data model parameters, WFA Data Elements and Multi AP parameters, WPA3 and 802.11ax support parameters, support for MQTT5.0, |

| Issue Number | Approval Date | Publication Date | Issue Editor | Changes |
|---|---|---|---|---|
| | | | | package capture diagnostics.<br><br>Defines version 2.13 of the Device data model (Device: 2.13). |

Comments or questions about this Broadband Forum Technical Report should be directed to info@broadband-forum.org.

**Editor:** Klaus Wich, Huawei

**Work Area Director(s):** Jason Walls, QA Cafe
John Blackford, ARRIS

**Project Stream Leader(s):** Klaus Wich, Huawei
William Lupton, Broadband Forum

# Table of Contents



## List of Figures



**List of Tables**



## Executive Summary

TR-181 Issue 2 defines version 2 of the Device data model (Device:2). The Device:2 data model applies to all types of TR-069 or USP enabled devices, including End Devices, Residential Gateways, and other Network Infrastructure Devices.

The Device:2 data model defined in this Technical Report consists of a set of data objects covering things like basic device information, time-of-day configuration, network interface and protocol stack configuration, routing and bridging management, throughput statistics, and diagnostic tests. It also defines a baseline profile that specifies a minimum level of data model support.

The cornerstone of the Device:2 data model is the interface stacking mechanism. Network interfaces and protocol layers are modeled as independent data objects that can be stacked, one on top of the other, into whatever configuration a device might support.

# 1 Purpose and Scope

## 1.1 Purpose

This Technical Report defines version 2 of the Device data model (Device:2). The Device:2 data model applies to all types of TR-069 or USP enabled devices, including End Devices, Residential Gateways, and other Network Infrastructure Devices.

## 1.2 Scope

The Device:2 data model defined in this Technical Report consists of a set of data objects covering things like basic device information, time-of-day configuration, network interface and protocol stack configuration, routing and bridging management, throughput statistics, and diagnostic tests. It also defines a baseline profile that specifies a minimum level of data model support.

The cornerstone of the Device:2 data model is the interface stacking mechanism. Network interfaces and protocol layers are modeled as independent data objects (a.k.a. interface objects) that can be stacked, one on top of the other, into whatever configuration a device might support.

Because the Device:2 data model can be used with either the USP or the CWMP (TR-069) protocol, it contains some objects and parameters which only apply if the specific protocol is used.

Figure 1 illustrates the top-level Device:2 data model structure for CWMP, Figure 2 the top-level Device:2 data model structure for USP.

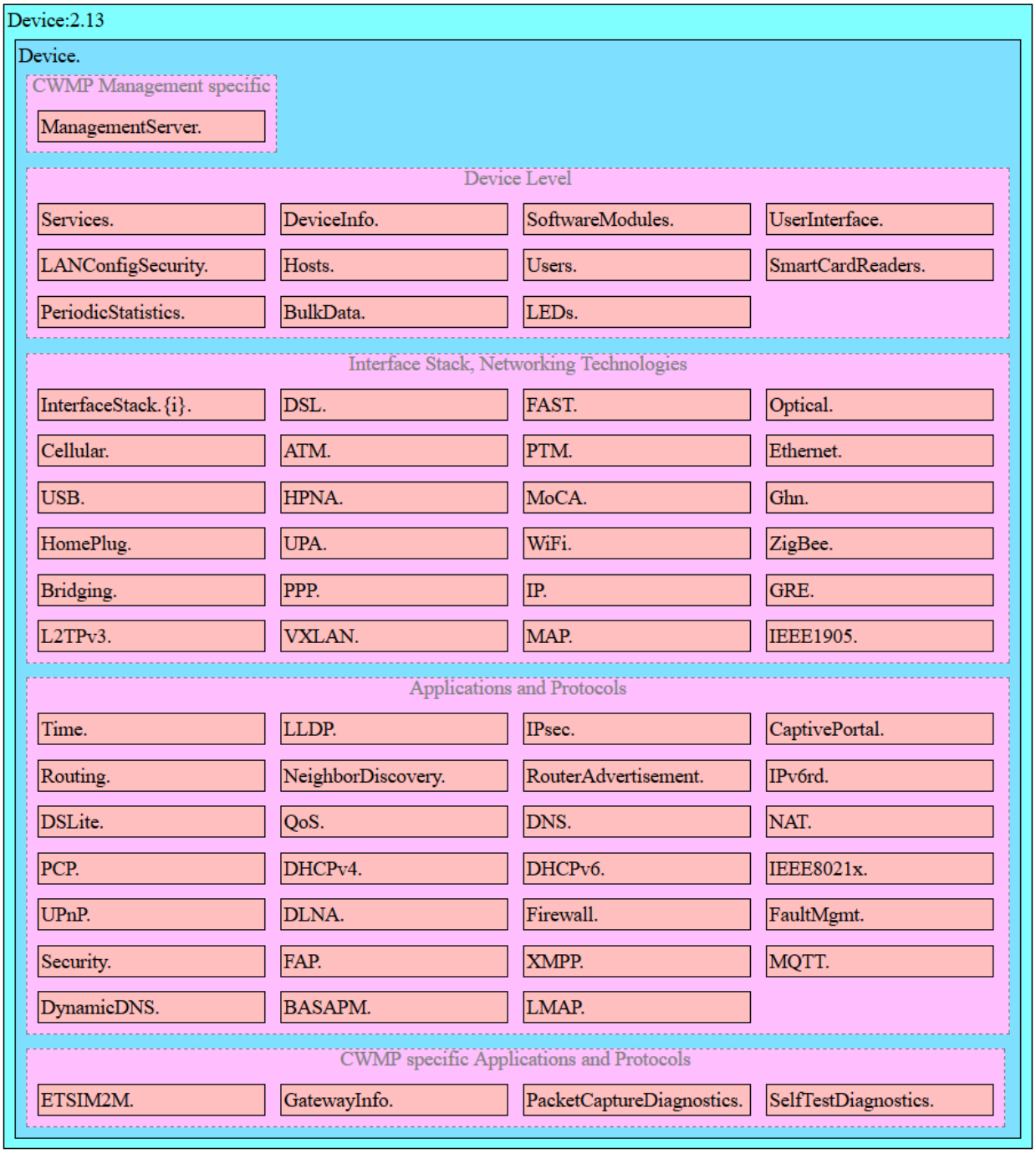



**Figure 1 – CWMP specific Device:2 Data Model Structure – Overview**

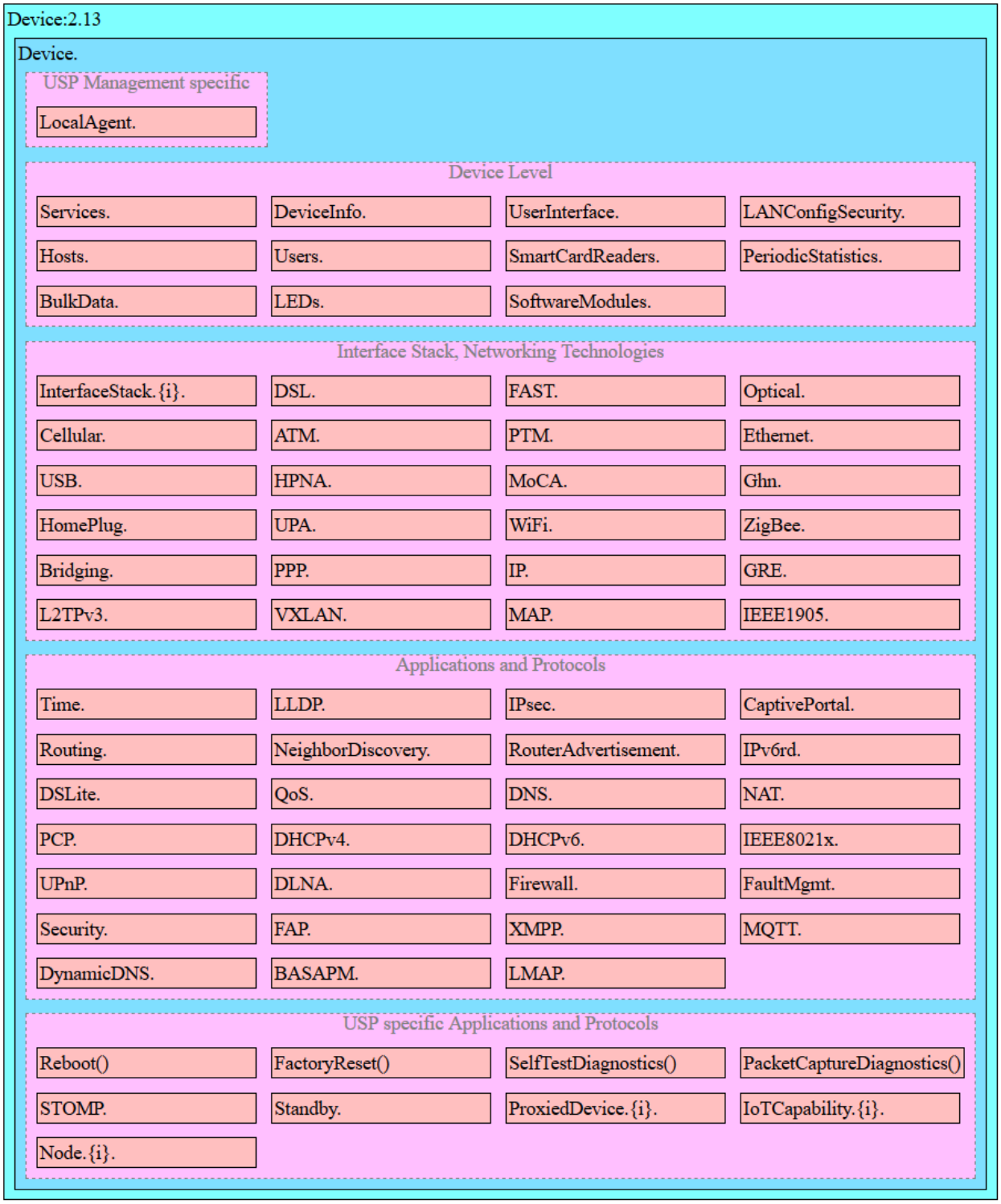



**Figure 2 – USP specific Device:2 Data Model Structure – Overview**

### 1.2.1 Detailed structure for common elements

The next figures illustrate the data model structure of the common parts in greater detail. This structure applies equally for USP and CWMP. See Section 5 for the complete list of objects.

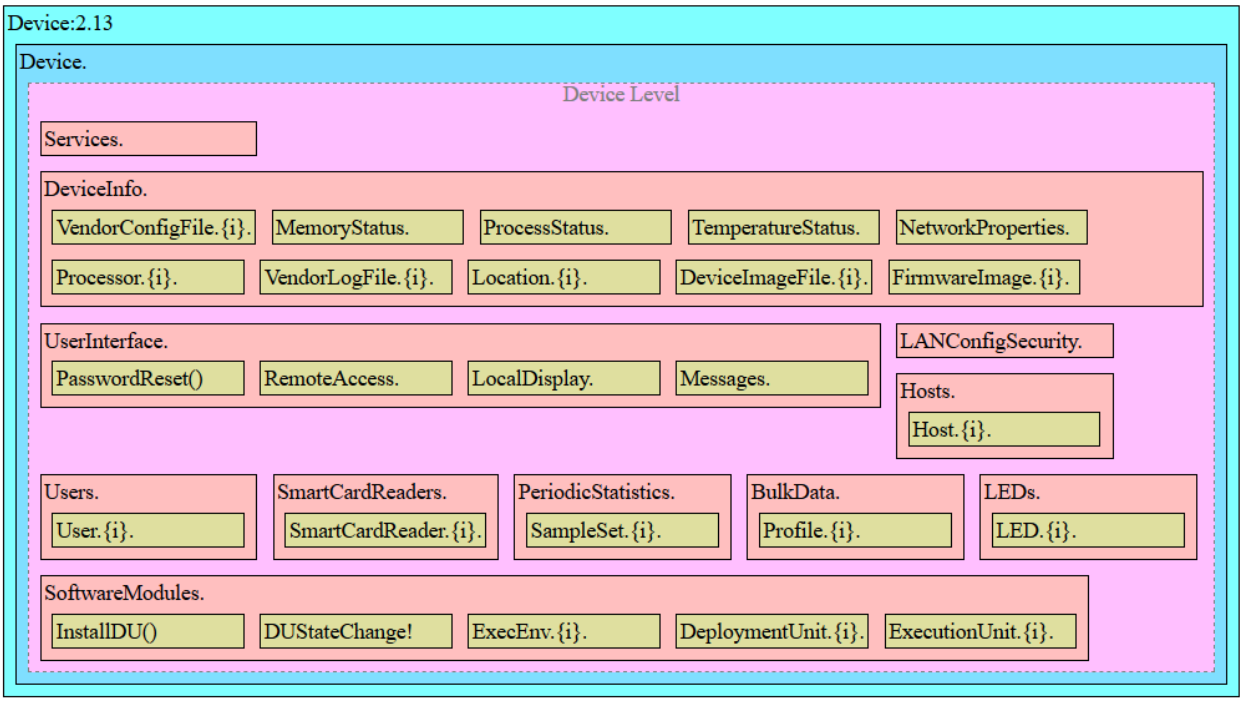



**Figure 3 – Device:2 Data Model Structure – Device Level**

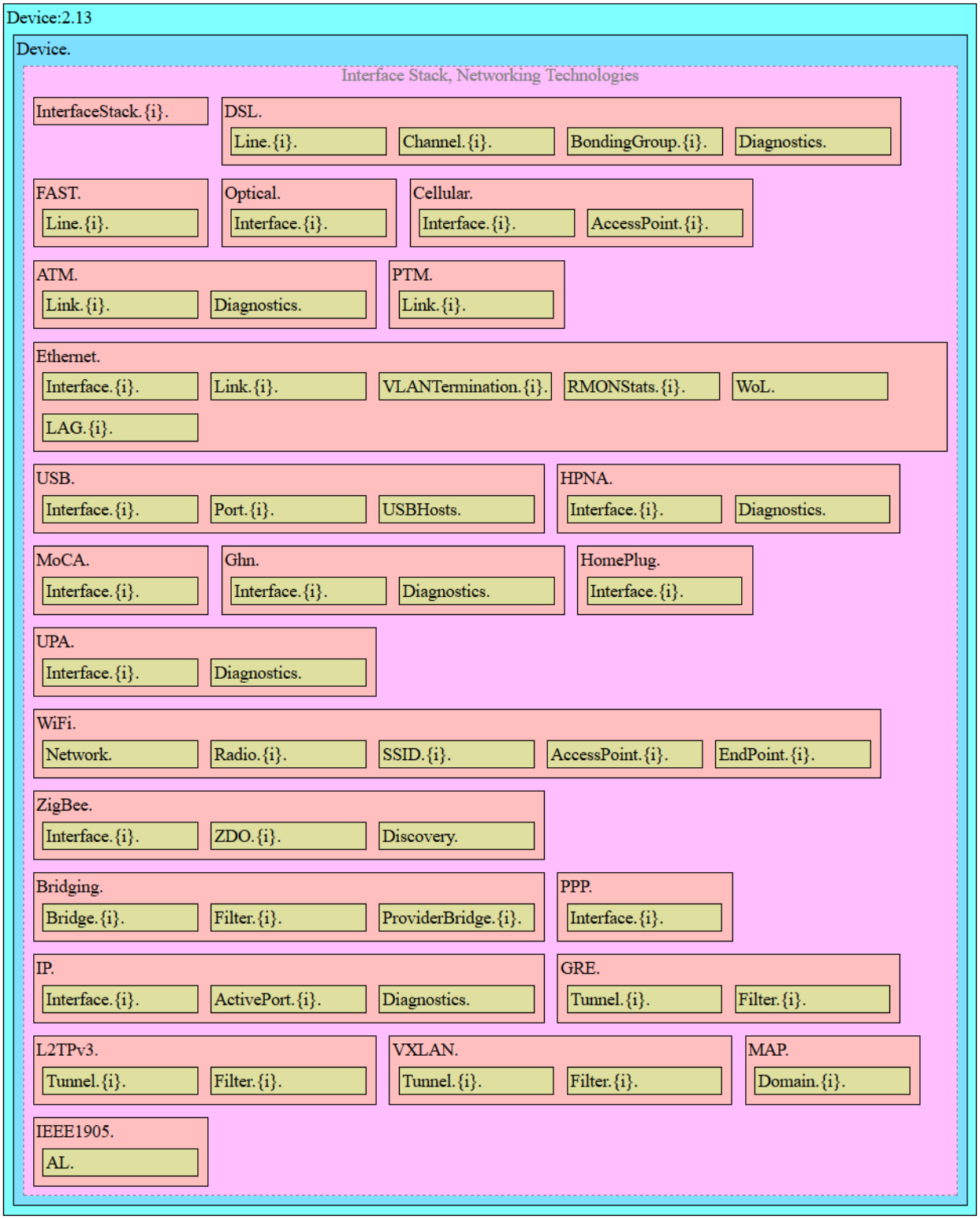



**Figure 4 – Device:2 Data Model Structure – Common Interface Stack and Networking Technologies**

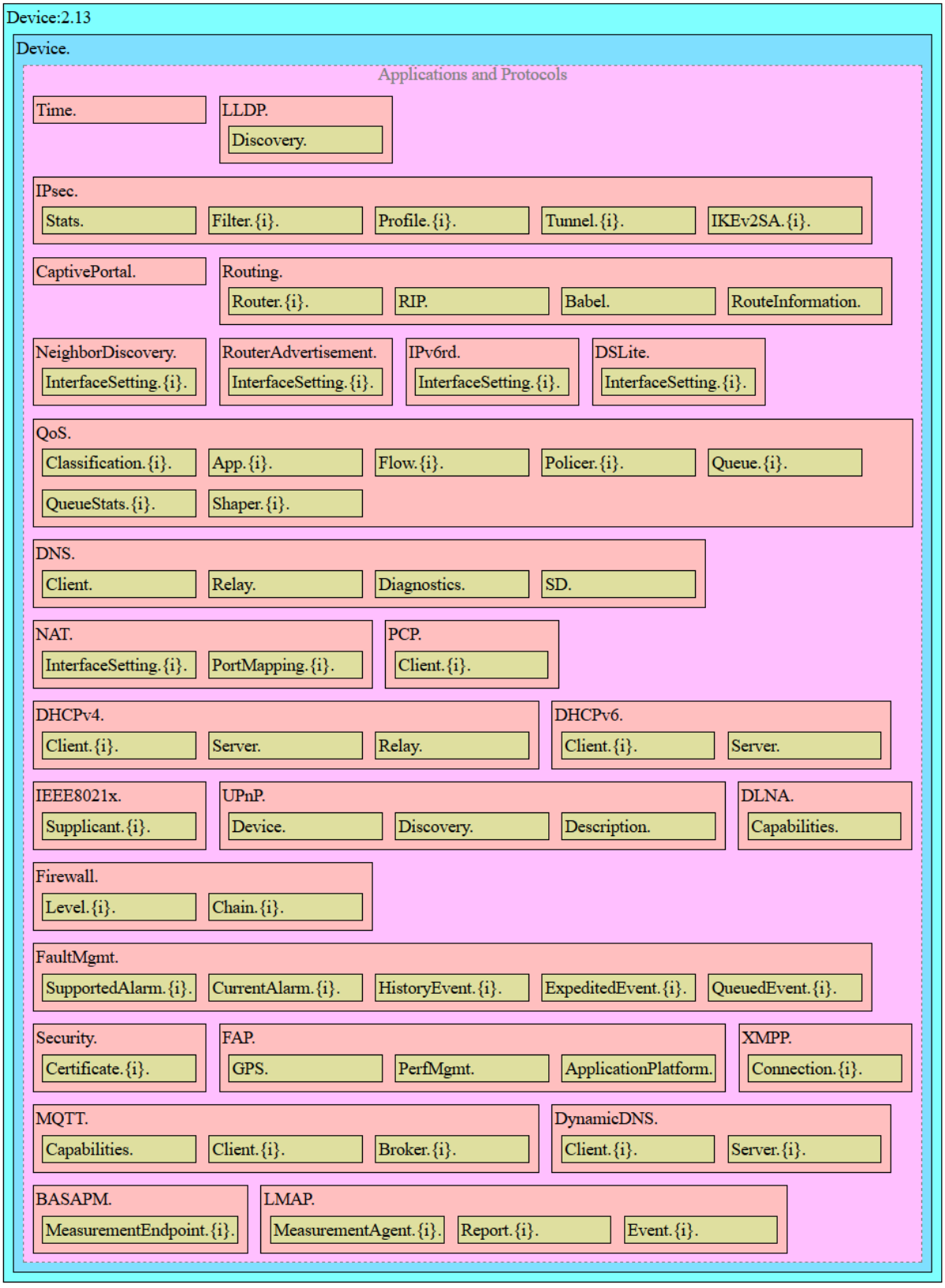


**Figure 5 – Device:2 Data Model Structure – Common Applications and Protocols**

### 1.2.2 Detailed structure for CWMP specific elements

The next figures illustrate the data model structure of the CWMP specific parts in greater detail. See Section 5 for the complete list of objects.

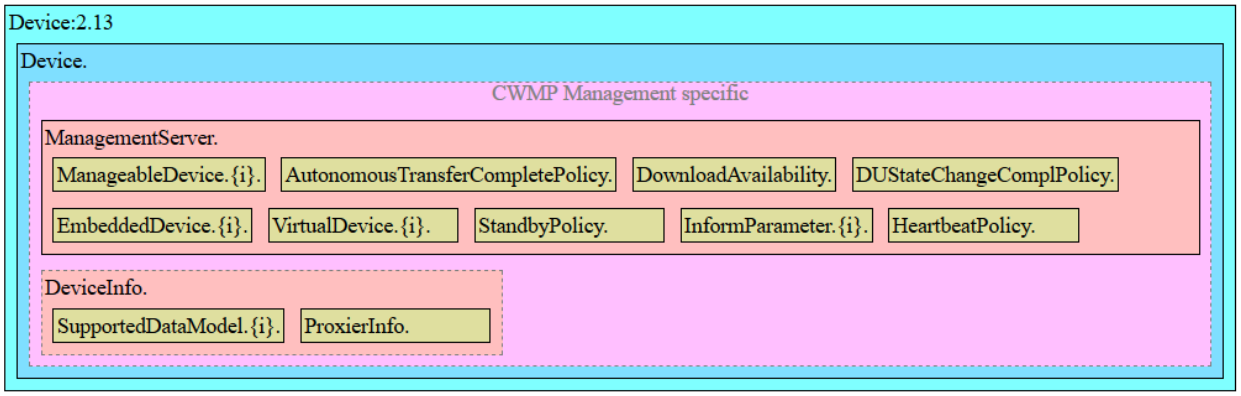



**Figure 6 – Device:2 Data Model Structure – CWMP Management**

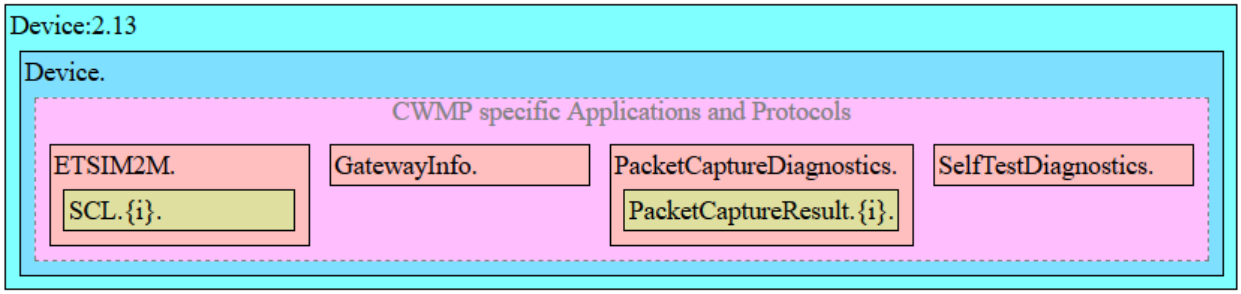



**Figure 7 – Device:2 Data Model Structure – CWMP specific applications and protocols**

### 1.2.3 Detailed structure for USP specific elements

The next figures illustrate the data model structure of the USP specific parts in greater detail. See Section 5 for the complete list of objects.

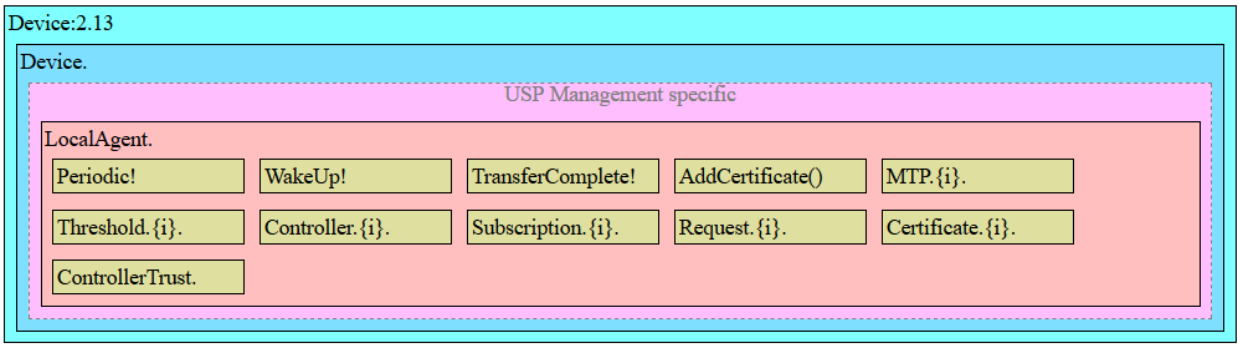



**Figure 8 – Device:2 Data Model Structure – USP Management**

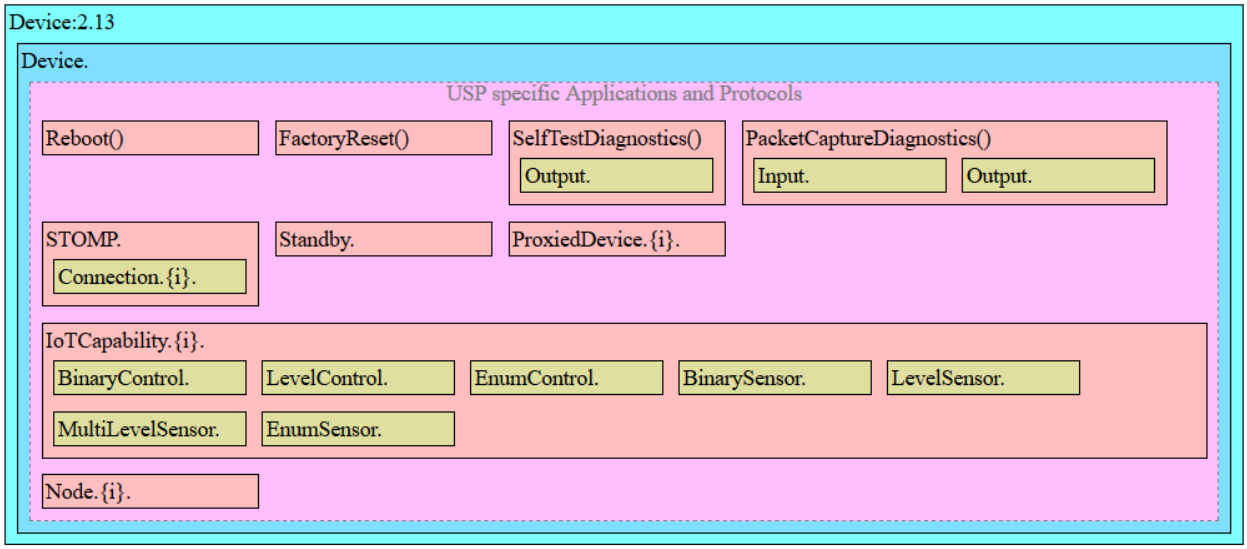



**Figure 9 – Device:2 Data Model Structure – USP specific applications and protocols**

# 2 References and Terminology

## 2.1 Conventions

In this Technical Report, several words are used to signify the requirements of the specification. These words are always capitalized. More information can be found be in RFC 2119 [1].

| | |
|---|---|
| **MUST** | This word, or the term "REQUIRED", means that the definition is an absolute requirement of the specification. |
| **MUST NOT** | This phrase means that the definition is an absolute prohibition of the specification. |
| **SHOULD** | This word, or the term "RECOMMENDED", means that there could exist valid reasons in particular circumstances to ignore this item, but the full implications need to be understood and carefully weighed before choosing a different course. |
| **SHOULD NOT** | This phrase, or the phrase "NOT RECOMMENDED" means that there could exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications need to be understood and the case carefully weighed before implementing any behavior described with this label. |
| **MAY** | This word, or the term "OPTIONAL", means that this item is one of an allowed set of alternatives. An implementation that does not include this option MUST be prepared to inter-operate with another implementation that does include the option. |

The key words "DEPRECATED" and "OBSOLETED" in this Technical Report are to be interpreted as defined in TR-106 [3].

## 2.2 References

The following references are of relevance to this Technical Report. At the time of publication, the editions indicated were valid. All references are subject to revision; users of this Technical Report are therefore encouraged to investigate the possibility of applying the most recent edition of the references listed below.

A list of currently valid Broadband Forum Technical Reports is published at www.broadband-forum.org.

[1] RFC 2119, *Key words for use in RFCs to Indicate Requirement Levels*, IETF, 1997

[2] TR-069 Amendment 6, *CPE WAN Management Protocol*, Broadband Forum, 2018

[3] TR-106 Amendment 8, *Data Model Template for CWMP Endpoints and USP Agents*, Broadband Forum, 2018

[4] RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, IETF, 2005

[5] *XML Schema Part 0: Primer Second Edition*, W3C, 2004

[6] RFC 2863, *The Interfaces Group MIB*, IETF, 2000

[7] X.200, Information technology - Open Systems Interconnection - Basic Reference Model: The basic model, ITU-T, 1994

[8] 802.1D-2004, Media Access Control (MAC) Bridges, IEEE, 2004

[9] 802.1Q-2011, Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks, IEEE, 2011

[10] RFC 2597, Assured Forwarding PHB Group, IETF, 1999

[11] RFC 3246, An Expedited Forwarding PHB (Per-Hop Behavior), IETF, 2002

[12] RFC 3261, SIP: Session Initiation Protocol, IETF, 2002

[13] RFC 3435, Media Gateway Control Protocol (MGCP) - Version 1.0, IETF, 2003

[14] RFC 4566, SDP: Session Description Protocol, IETF, 2006

[15] RFC 2453, RIP Version 2, IETF, 1998

[16] RFC 2460, Internet Protocol Version 6 (IPv6) Specification, IETF, 1998

[17] RFC 2464, Transmission of IPv6 Packets over Ethernet Networks, IETF, 1998

[18] RFC 3315, Dynamic Host Configuration Protocol for IPv6 (DHCPv6), IETF, 2003

[19] RFC 3633, IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6, IETF, 2003

[20] RFC 4191, Default Router Preferences and More-Specific Routes, IETF, 2005

[21] RFC 4193, Unique Local IPv6 Unicast Addresses, IETF, 2005

[22] RFC 4861, Neighbor Discovery for IP version 6 (IPv6), IETF, 2007

[23] RFC 4862, IPv6 Stateless Address Autoconfiguration, IETF, 2007

[24] RFC 5072, IP Version 6 over PPP, IETF, 2007

[25] RFC 5969, IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) – Protocol Specification, IETF, 2010

[26] RFC 6106, IPv6 Router Advertisement Options for DNS Configuration, IETF, 2010

[27] RFC 6333, Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion, IETF, 2011

[28] RFC 6334, Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Options for Dual-Stack Lite, IETF, 2011

[29] TR-101 Issue 2, Migration to Ethernet Based DSL Aggregation, Broadband Forum, 2011

[30] TR-124 Issue 2, Functional Requirements for Broadband Residential Gateway Devices, Broadband Forum, 2010

[31] TR-177 Corrigendum 1, IPv6 in the context of TR-101, Broadband Forum, 2017

[32] TR-187 Issue 2, IPv6 for PPP Broadband Access, Broadband Forum, 2013

[33] ICSA Baseline Modular Firewall Certification Criteria, Baseline module – version 4.1, ICSA Labs, 2008

[34] ICSA Residential Modular Firewall Certification Criteria, Required Services Security Policy – Residential Category module – version 4.1, ICSA Labs, 2008

[35] RFC 4301, Security Architecture for the Internet Protocol, IETF, 2005

[36] RFC 4302, IP Authentication Header (AH), IETF, 2005.

[37] RFC 4303, IP Encapsulating Security Payload (ESP), IETF, 2005

[38] RFC 5996, Internet Key Exchange Protocol Version 2 (IKEv2), IETF, 2010

[39] ETSI TS 102 690 v1.2.1, Machine-to-Machine Communications (M2M Functional Architecture), ETSI, 2013

[40] ETSI TS 102 921 v1.3.1, M2M mIa, dIa and mId Interfaces, ETSI, 2014

[41] ETSI TS 103 093 v1.2.1, Machine to Machine (M2M); BBF TR-069 Compatible Data Model for ETSI M2M, ETSI, 2012

[42] ZigBee-2007, ZigBee Specification, The ZigBee Alliance, 2007

[43] RFC 6887, Port Control Protocol (PCP), IETF, 2013

[44] RFC 6970, Universal Plug and Play (UPnP) Internet Gateway Device – Port Control Protocol Interworking Function (IGD-PCP IWF), IETF, 2013

[45] RFC 7291, DHCP Options for the Port Control Protocol (PCP), IETF, 2014

[46] RFC 7648, Port Control Protocol (PCP) Proxy Function, IETF, 2015

[47] RFC 7488, PCP Server Selection, IETF, 2014

[48] draft-boucadair-pcp-flow-examples, Port Control Protocol (PCP) Flow Examples, IETF, 2013

[49] RFC 2661, Layer Two Tunneling Protocol "L2TP", IETF, 1999

[50] RFC 2784, Generic Routing Encapsulation (GRE), IETF, 2000

[51] RFC 2890, Key and Sequence Number Extensions to GRE, IETF, 2000

[52] RFC 7597, Mapping of Address and Port with Encapsulation (MAP), IETF, 2014

[53] RFC 7598, DHCPv6 Options for configuration of Softwire Address and Port Mapped Clients, IETF, 2014

[54] RFC 7599, Mapping of Address and Port using Translation (MAP-T), IETF, 2014

[55] TR-059, DSL Evolution - Architecture Requirements for the Support of QoS-Enabled IP Services, Broadband Forum, 2013

[56] RFC 4119, A Presence-based GEOPRIV Location Object Format, IETF, 2005

[57] RFC 5491, GEOPRIV Presence Information Data Format Location Object (PIDF-LO) Usage Clarification, Considerations, and Recommendation, IETF, 2009

[58] RFC 5139, Revised Civic Location Format for Presence Information Data Format Location Object (PIDF-LO), IETF, 2008

[59] RFC 4479, A Data Model for Presence, IETF, 2006

[60] IANA Method Tokens, Method Tokens, IANA, 2008

[61] TR-143 Amendment 1 Corrigendum 1, Enabling Network Throughput Performance Tests and Statistical Monitoring, Broadband Forum, 2014

[62] TR-232, Bulk Data Collection, Broadband Forum, 2012

[63] TR-304, Broadband Access Service Attributes and Performance Metrics, Broadband Forum, 2015

[64] RFC 7594, A Framework for Large-Scale Measurement of Broadband Performance (LMAP), IETF, 2015

[65] RFC 8193, Information Model for Large-Scale Measurement Platforms, IETF, 2017

[66] RFC 4122, A Universally Unique IDentifier (UUID) URN Namespace, IETF, 2005

[67] TR-369, User Services Platform (USP), Broadband Forum, 2018

## 2.3 Definitions

The following terminology is used throughout this Technical Report.

| | |
|---|---|
| **ACS** | Auto-Configuration Server. This is a component in the broadband network responsible for *CWMP* auto-configuration of the *CPE* for advanced services. |
| **Agent** | A generic term that refers (as appropriate) to either a CWMP Endpoint or to a USP Agent. |
| **CPE** | Customer Premises Equipment; refers (as appropriate) to any *CWMP*-enabled [2] or *USP*-enabled [67] device and therefore covers both Internet Gateway devices and LAN-side end devices. |
| **Command** | A named element allowing a USP Controller to execute an operation on a USP Agent. This concept does not apply to CWMP, which uses Objects and/or Parameters to simulate operations. |
| **Component** | A named collection of *Objects* and/or *Parameters* and/or Profiles that can be included anywhere within a *Data Model*. |
| **Controller** | A generic term that refers (as appropriate) to either a CWMP ACS or a USP Controller. |
| **CWMP** | *CPE* WAN Management Protocol. Defined in TR-069 [2], CWMP is a communication protocol between an *ACS* and a CWMP-enabled *CPE* that defines a mechanism for secure auto-configuration of a *CPE* and other *CPE* management functions in a common framework. |
| **CWMP Endpoint** | A CWMP termination point used by a CWMP-enabled CPE for communication with the ACS. |
| **Data Model** | A hierarchical set of *Objects, Parameters, Commands and/or Events* that define the managed objects accessible via a particular *Agent*. |
| **Device** | Used here as a synonym for *CPE*. |
| **DM Instance** | Data Model Schema instance document. This is an XML document that conforms to the *DM Schema* and to any additional rules specified in or referenced by the *DM Schema*. |
| **DM Schema** | Data Model Schema. This is the XML Schema [5] that is used for defining data models for use with *CWMP and USP*. |
| **Downstream Interface** | A physical interface object whose Upstream parameter is set to *false*, or an interface that is associated with such a physical interface via the InterfaceStack. For example, a downstream IP Interface is an IP.Interface object that is associated with an Upstream=false physical layer interface. |
| **Event** | An indication that something of interest has happened that requires the Agent to notify the Controller. |
| **Interface Object** | A type of *Object* that models a network interface or protocol layer. Commonly referred to as an interface. They can be stacked, one on top of the other, using *Path References* in order to dynamically define the relationships between interfaces. |
| **Object** | An internal node in the name hierarchy, i.e., a node that can have Object, Parameter, Command and/or Event children. An Object name is a Path Name. |

| | |
|---|---|
| **Parameter** | A name-value pair that represents part of a CPE or USP Agent's configuration or status. A Parameter name is a Path Name. |
| **Path Name** | A name that has a hierarchical structure similar to files in a directory, with each level separated by a "." (dot). References an Object, Parameter, Command or Event. |
| **Path Reference** | Describes how a parameter can reference another parameter or object via its path name (Section A.2.3.4/TR-106 [3]). Such a reference can be weak or strong (Section A.2.3.6/TR-106 [3]). |
| **Upstream Interface** | A physical interface object whose Upstream parameter is set to *true*, or an interface that is associated with such a physical interface via the InterfaceStack. For example, an upstream IP Interface is an IP.Interface object that is associated with an Upstream=true physical layer interface. |
| **USP** | User Services Platform. Defined in TR-369 [67], USP is an evolution of CWMP that allows applications to manipulate Service Elements in a network of Controllers and Agents. |
| **USP Agent** | A USP Agent is a USP Endpoint that exposes Service Elements to one or more USP Controllers. |
| **USP Controller** | A USP Controller is a USP Endpoint that manipulates Service Elements through one or more USP Agents. |
| **USP Endpoint** | A USP Endpoint is a termination point for a USP message. |

## 2.4 Abbreviations

This Technical Report uses the following abbreviations:

| | |
|---|---|
| ATM | Asynchronous Transfer Mode |
| CGN | Carrier Grade NAT |
| DHCP | Dynamic Host Configuration Protocol |
| DSL | Digital Subscriber Line |
| IP | Internet Protocol |
| IPsec | Internet Protocol Security |
| M2M | Machine to Machine |
| NAT | Network Address Translation |
| NSCL | Network Service Capability Layer |
| OSI | Open Systems Interconnection |
| PCP | Port Control Protocol |
| PPP | Point-to-Point Protocol |
| PTM | Packet Transfer Mode |
| REM | Remote Entity Management |
| RG | Residential Gateway |
| RPC | Remote Procedure Call |
| SCL | Service Capability Layer |
| SSID | Service Set Identifier |
| TR | Technical Report |
| URI | Uniform Resource Identifier [4] |
| URL | Uniform Resource Locator [4] |
| USB | Universal Serial Bus |
| UUID | Universally Unique IDentifier |
| WFA | Wi-Fi Alliance |
| xREM | x (Device or Gateway) Remote Entity Management |
| ZDO | ZigBee Device Object |

# 3 Technical Report Impact

## 3.1 Energy Efficiency

TR-181 Issue 2 Amendment 13 has no impact on Energy Efficiency.

## 3.2 IPv6

TR-181 Issue 2 Amendment 13 defines IPv6 extensions[1] to the Device:2 data model.

## 3.3 Security

TR-181 Issue 2 Amendment 13 has no impact on Security.

## 3.4 Privacy

TR-181 Issue 2 Amendment 13 has no impact on Privacy.

[1] *Introduced in Issue 2 Amendment 2*

# 4 Architecture

## 4.1 Interface Layers

This Technical Report models network interfaces and protocol layers as independent data objects, generally referred to as interface objects (or interfaces). Interface objects can be stacked, one on top of the other, using path references in order to dynamically define the relationships between interfaces.

The interface object and interface stack are concepts inspired by RFC 2863 [6].

Within the Device:2 data model, interface objects are arbitrarily restricted to definitions that operate at or below the IP network layer (i.e. layers 1 through 3 of the OSI model [7]). However, vendor-specific interface objects MAY be defined which fall outside this restricted scope.

Figure 10 lists the interface objects defined in the Device:2 data model. The indicated OSI layer is non-normative; it serves as a guide only, illustrating at what level in the stack an interface object is expected to appear. However, a CPE need not support or use all interfaces, which means that the figure does not reflect all possible stacking combinations and restrictions. For example, one CPE stack might exclude DSL Bonding, while another CPE stack might include DSL Bonding but exclude Bridging, while still another might include VLANTermination under PPP, or VLANTermination under IP with no PPP, or even Ethernet Link under IP with no VLANTermination and no PPP.

*NOTE – Throughout this Technical Report, object names are often abbreviated in order to improve readability. For example, Device.Ethernet.VLANTermination.{i}. is the full name of a Device:2 object, but might casually be referred to as Ethernet.VLANTermination.{i} or VLANTermination.{i} or VLANTermination, just so long as the abbreviation is unambiguous (with respect to similarly named objects defined elsewhere within the data model).*

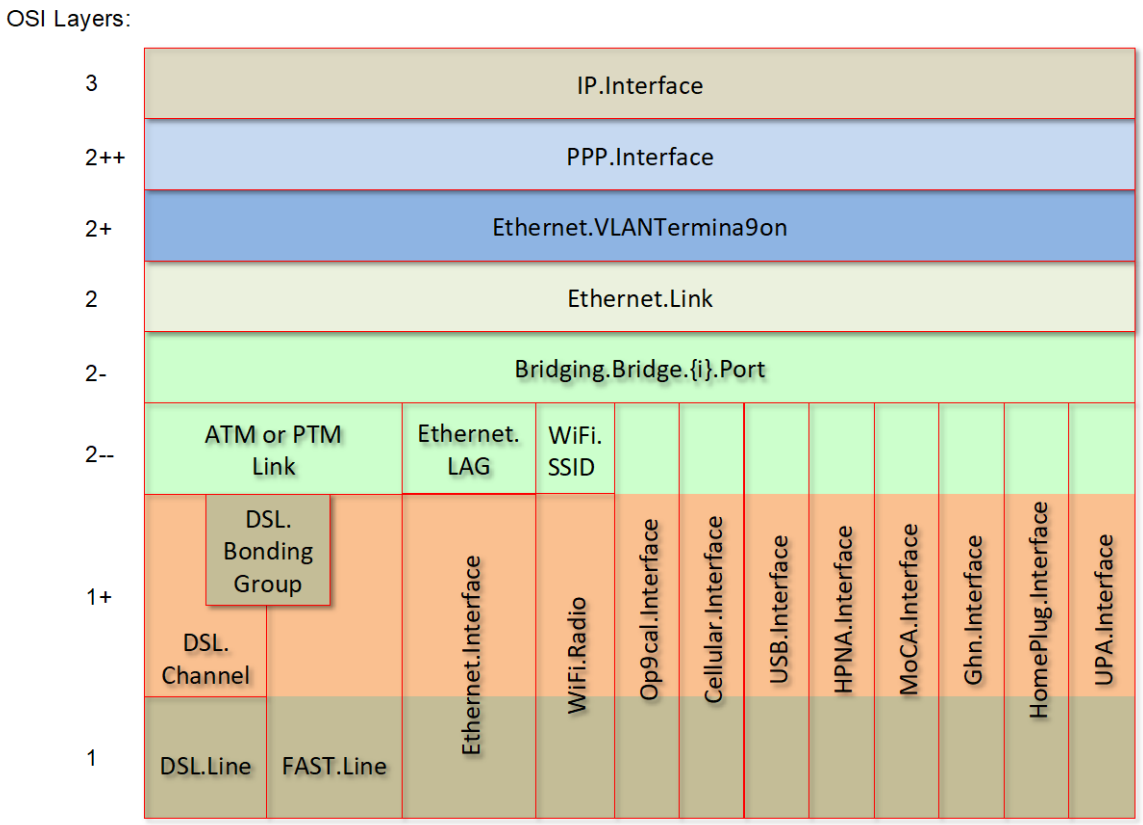



**Figure 10 – OSI Layers and Interface Objects**[2] [3]

## 4.2 Interface objects

An interface object is a type of network interface or protocol layer. Each type of interface is modeled by a Device:2 data model table, with a row per interface instance (e.g. IP.Interface.{i} for IP Interfaces).

Each interface object contains a core set of parameters and objects, which serves as the template for defining interface objects within the data model. Interface objects can also contain other parameters and sub-objects specific to the type of interface.

[2] *Note that, because new minor versions of the Device:2 data model can be defined without re-publishing this document, the figure is not necessarily up-to-date.*

[3] *The Bridge.{i}.Port.{i} object models both management (upwards facing) Bridge Ports and non-management (downwards facing) Bridge Ports, where each instance is configured as one or the other. Management Bridge Ports are stacked above non-management Bridge Ports.*

The core set of parameters consists of:

- Enable — The administrative state of the interface (i.e. boolean indicating enabled or disabled)
- Status — The operational state of the interface (i.e. Up, Down, Unknown, Dormant, NotPresent, LowerLayerDown, Error)
- Alias — An alternate name used to identify the interface, which is assigned an initial value by the CPE but can later be chosen by the Controller
- Name — The textual name used to identify the interface, which is chosen by the CPE
- LastChange — The accumulated time in seconds since the interface entered its current operational state
- LowerLayers — A list of path references to interface objects that are stacked immediately below the interface

Also, a core set of statistics parameters is contained within a Stats sub-object. The definition of these parameters MAY be customized for each interface type. The core set of parameters within the Stats sub-object consists of:

- BytesSent — The total number of bytes transmitted out of the interface, including framing characters.
- BytesReceived — The total number of bytes received on the interface, including framing characters.
- PacketsSent — The total number of packets transmitted out of the interface.
- PacketsReceived — The total number of packets received on the interface.
- ErrorsSent — The total number of outbound packets that could not be transmitted because of errors.
- ErrorsReceived — The total number of inbound packets that contained errors preventing them from being delivered to a higher-layer protocol.
- UnicastPacketsSent — The total number of packets requested for transmission, which were not addressed to a multicast or broadcast address at this layer, including those that were discarded or not sent.
- UnicastPacketsReceived — The total number of received packets, delivered by this layer to a higher layer, which were not addressed to a multicast or broadcast address at this layer.
- DiscardPacketsSent — The total number of outbound packets, which were chosen to be discarded even though no errors had been detected to prevent their being transmitted.

| | |
|---|---|
| • DiscardPacketsReceived | The total number of inbound packets, which were chosen to be discarded even though no errors had been detected to prevent their being delivered. |
| • MulticastPacketsSent | The total number of packets that higher-layer protocols requested for transmission and which were addressed to a multicast address at this layer, including those that were discarded or not sent. |
| • MulticastPacketsReceived | The total number of received packets, delivered by this layer to a higher layer, which were addressed to a multicast address at this layer. |
| • BroadcastPacketsSent | The total number of packets that higher-level protocols requested for transmission and which were addressed to a broadcast address at this layer, including those that were discarded or not sent. |
| • BroadcastPacketsReceived | The total number of received packets, delivered by this layer to a higher layer, which were addressed to a broadcast address at this layer. |
| • UnknownProtoPackets-Received | The total number of packets received via the interface, which were discarded because of an unknown or unsupported protocol. |

*NOTE – The CPE MUST reset an interface's Stats parameters (unless otherwise stated in individual object or parameter descriptions) either when the interface becomes operationally down due to a previous administrative down (i.e. the interface's Status parameter transitions to a down state after the interface is disabled) or when the interface becomes administratively up (i.e. the interface's Enable parameter transitions from false to true). Administrative and operational status is discussed in Section 4.2.2.*

### 4.2.1 Lower Layers

Each interface object can be stacked on top of zero or more other interface objects, which MUST be specified using its LowerLayers parameter. By having each interface object, in turn, reference the interface objects in its lower layer; a logical hierarchy of all interface relationships is built up.

The LowerLayers parameter is a comma-separated list of path references to interface objects. Each item in the list represents an interface object that is stacked immediately below the referencing interface. If a referenced interface is deleted, the CPE MUST remove the corresponding item from this list (i.e. items in the LowerLayers parameter are strong references).

These relationships between interface objects can either be set by management action, in order to specify new interface configurations, or be pre-configured within the CPE.

A CPE MUST reject any attempt to set LowerLayers values that would result in an invalid or unsupported configuration. The corresponding fault response from the CPE MUST indicate this, using the appropriate protocol response.

The lowest layer in a fully configured and operational stack is generally the physical interface (e.g. DSL Line instance representing a DSL physical link). Within these physical interface objects the LowerLayers parameter will be an empty list, unless some lower layer vendor-specific interface objects are defined and present. Higher layer interface objects MAY operate without a physical layer being modeled, however this is a local matter to the CPE.

Figure 11 illustrates the use of the LowerLayers parameter. A, B, C, and D represent interface objects. Interface A's LowerLayers parameter references interfaces B and C. Interface B's LowerLayers parameter references interface D. Interfaces C and D have no interface references specified in their LowerLayers parameters. In this way, a multi-layered interface stack is configured. If the Controller were to delete interface B, then the CPE would update interface A's LowerLayers parameter to no longer reference interface B (and interface D would be stranded, no longer referenced by the now deleted interface B).

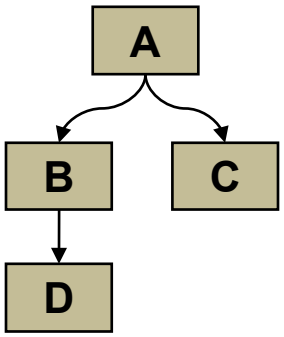



**Figure 11 – Interface LowerLayers**

### 4.2.2 Administrative and Operational Status

*NOTE – Many of the requirements outlined in this section were derived from Section 3.1.13/RFC 2863 [6].*

An interface object's Enable and Status parameters specify the current administrative and operational status of the interface, respectively. Valid values for the Status parameter are: Up, Down, Unknown, Dormant, NotPresent, LowerLayerDown, and Error.

The CPE MUST do everything possible in order to follow the operational state transitions as described below. In some cases, these requirements are defined as SHOULD; this is not an indication that they are optional. These transitions, and the relationship between the Enable parameter and the Status parameter, are required behavior – it is simply the timing of how long these state transitions take that is implementation specific.

When the Enable parameter is *false* the Status parameter SHOULD normally be *Down (or NotPresent or Error if there is a fault condition on the interface)*. Note that when the Enable parameter transitions to *false*, it is possible that the Status parameter's transition to *Down* might occur after a small time lag if the CPE needs to first complete certain operations (e.g. finish transmitting a packet).

When the Enable parameter is changed to *true*, the Status SHOULD do one of the following:

- Change to *Up* if and only if the interface is able to transmit and receive network traffic.
- Change to *Dormant* if and only if the interface is operable, but is waiting for external actions before it can transmit and receive network traffic.
- Change to *LowerLayerDown* if and only if the interface is prevented from entering the *Up* state because one or more of the interfaces beneath it is down.
- Remain in the *Error* state if there is an error or other fault condition detected on the interface.
- Remain in the *NotPresent* state if the interface has missing (typically hardware) components.
- Change to *Unknown* if the state of the interface cannot be determined for some reason.

The *Dormant* state indicates that the interface is operable, but it is waiting for external events to occur before it can transmit/receive traffic. When such events occur, and the interface is then able to transmit/receive traffic, the Status SHOULD change to the *Up* state. Note that both the *Up* and *Dormant* states are considered healthy states.

The *Down*, *NotPresent*, *LowerLayerDown,* and *Error* states all indicate that the interface is down. The *NotPresent* state indicates that the interface is down specifically because of a missing (typically hardware) component. The *LowerLayerDown* state indicates that the interface is stacked on top of one or more other interfaces, and that this interface is down specifically because one or more of these lower-layer interfaces is down.

The *Error* state indicates that the interface is down because an error or other fault condition was detected on the interface.

### 4.2.3 Stacking and Operational Status

*NOTE – The requirements outlined in this section were derived from Section 3.1.14/RFC 2863 [6].*

When an interface object is stacked on top of lower-layer interfaces (i.e. is not a bottommost layer in the stack), then:

- The interface SHOULD be *Up* if it is able to transmit/receive traffic due to one or more interfaces lower down in the stack being *Up*, irrespective of whether other interfaces below it are in a non-*Up* state (i.e. the interface is functioning in conjunction with at least some of its lower-layered interfaces).
- The interface MAY be *Up* or *Dormant* if one or more interfaces lower down in the stack are *Dormant* and all other interfaces below it are in a non-*Up* state.
- The interface is expected to be *LowerLayerDown* while all interfaces lower down in the stack are either *Down*, *NotPresent*, *LowerLayerDown,* or *Error*.

### 4.2.4 Vendor-specific Interface Objects

Vendor-specific interface objects MAY be defined and used. If such objects are specified by vendors, they MUST be preceded by *X_<VENDOR>_* and follow the syntax for vendor extensions used for parameter names (as defined in Section 3.3/TR-106 [3]).

If the Controller encounters an unknown vendor-specific interface object within a CPE's interface stack, rather than responding with a fault, the Controller MUST proceed as if this object's upper-layer interfaces were directly linked to its lower-layer interfaces. This applies whether the Controller encounters such an object via the InterfaceStack table (Section 4.3) or via an interface object's LowerLayers parameter.

Figure 12 illustrates a stacked vendor-specific interface object being bypassed by the Controller, where there is just one object below the vendor-specific object.

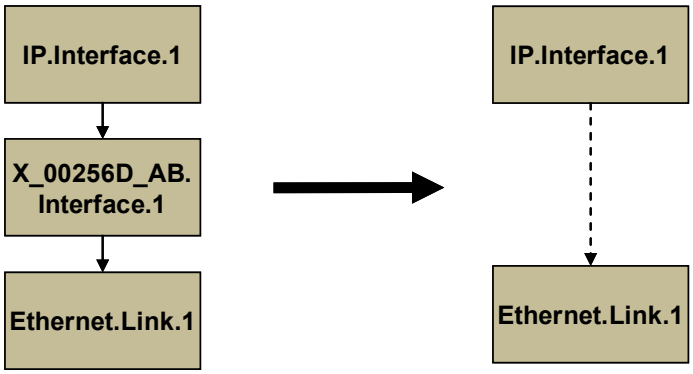



**Figure 12 – Ignoring a Vendor-specific Interface Object in the Stack**

Figure 13 illustrates a stacked vendor-specific interface object being bypassed by the Controller, where there are multiple objects below the vendor-specific object.

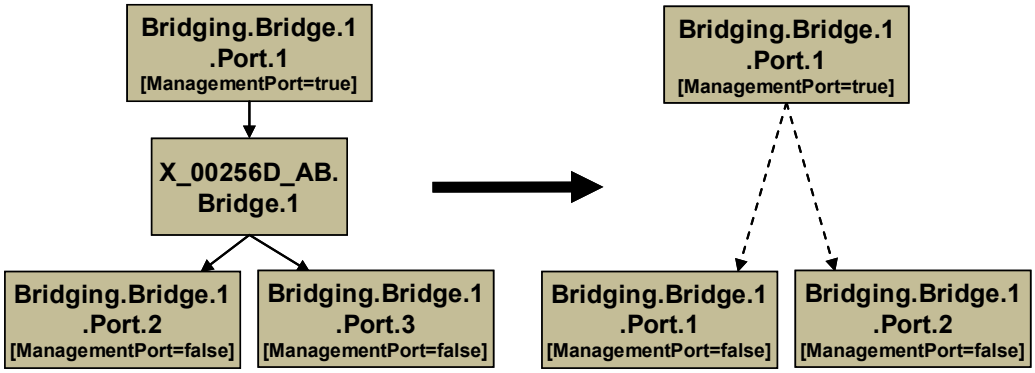



**Figure 13 – Ignoring a Vendor-specific Interface Object in the Stack (multiple sub-objects)**

## 4.3 InterfaceStack Table

Although the interface stack can be traversed via LowerLayers parameters (as described in Section 4.2.1 *Lower Layers*), an alternate mechanism is provided to aid in visualizing the overall stacking relationships and to quickly access objects within the stack.

The InterfaceStack table is a Device:2 data model object, namely *Device.InterfaceStack.{i}*. This is a read-only table whose rows are auto-generated by the CPE based on the current relationships that are configured between interface objects (via each interface instance's LowerLayers

parameter). Each table row represents a "link" between a higher-layer interface object (referenced by its HigherLayer parameter) and a lower-layer interface object (referenced by its LowerLayer parameter). This means that an InterfaceStack table row's HigherLayer and LowerLayer parameters will always both be non-null.

*NOTE – As a consequence, interface instances that have been stranded will not be represented within the InterfaceStack table[4]. It is also likely that multiple, disjoint groups of stacked interface objects will coexist within the table (for example, each IP interface will be the root of a disjoint group; unused "fragments", e.g. a secondary DSL channel with a configured ATM PVC that isn't attached to anything above, will linger if they remain interconnected; and finally, partially configured "fragments" can be present when an interface stack is being set up).*

A CPE MUST autonomously add or remove rows in the InterfaceStack table in response to the following circumstances:

- An interface's LowerLayers parameter was updated to remove a reference to another interface (i.e. a "link" is being removed from the stack).
- An interface's LowerLayers parameter was updated to add a reference to another interface (i.e. a "link" is being added to the stack).
- An interface was deleted that had referenced, or been referenced by, one other interface (i.e. a "link" is being removed from the stack).
- An interface was deleted that had referenced, or been referenced by, multiple interfaces (i.e. multiple "links" are being removed from the stack).

Once the CPE issues the response to the Controller request, all autonomous InterfaceStack table changes associated with the corresponding request (as described in the preceding paragraph) MUST be available for subsequent commands to operate on, regardless of whether or not these changes have been applied by the CPE.

As an example, Table 1 lists an InterfaceStack table configuration imagined for a fictitious, simple router. Each row in this table corresponds to a row in the InterfaceStack table. The specified objects and instance numbers are manufactured for the sake of this example; real world configurations will likely differ.

**Table 1 – Simple Router Example (InterfaceStack table)**

| Row/Instance | Higher Layer Interface | Lower Layer Interface |
|---|---|---|
| 1 | Device.IP.Interface.1 | Device.PPP.Interface.1 |
| 2 | Device.PPP.Interface.1 | Device.Ethernet.Link.1 |
| 3 | Device.Ethernet.Link.1 | Device.ATM.Link.1 |
| 4 | Device.ATM.Link.1 | Device.DSL.Channel.1 |
| 5 | Device.DSL.Channel.1 | Device.DSL.Line.1 |

[4] An interface instance is considered stranded when it has no lower layer references to or from other interface instances. Stranded interface instances will be omitted from the InterfaceStack table until such time as they are stacked, above or below another interface instance, via a LowerLayers parameter reference.

| Row/Instance | Higher Layer Interface | Lower Layer Interface |
|---|---|---|
| 6 | Device.IP.Interface.2 | Device.Ethernet.Link.2 |
| 7 | Device.Ethernet.Link.2 | Device.ATM.Link.2 |
| 8 | Device.ATM.Link.2 | Device.DSL.Channel.1 |
| 9 | Device.IP.Interface.3 | Device.Ethernet.Link.3 |
| 10 | Device.Ethernet.Link.3 | Device.Bridging.Bridge.1.Port.1 |
| 11 | Device.Bridging.Bridge.1.Port.1 | Device.Bridging.Bridge.1.Port.2 |
| 12 | Device.Bridging.Bridge.1.Port.2 | Device.Ethernet.Interface.1 |
| 13 | Device.Bridging.Bridge.1.Port.1 | Device.Bridging.Bridge.1.Port.3 |
| 14 | Device.Bridging.Bridge.1.Port.3 | Device.Ethernet.Interface.2 |
| 15 | Device.Bridging.Bridge.1.Port.1 | Device.Bridging.Bridge.1.Port.4 |
| 16 | Device.Bridging.Bridge.1.Port.4 | Device.WiFi.SSID.1 |
| 17 | Device.WiFi.SSID.1 | Device.WiFi.Radio.1 |

By looking at the rows from the example InterfaceStack table as a whole, we can visualize the overall stack configuration. Figure 14 shows how this information can be pictured. Interface instances are represented by colored boxes, while InterfaceStack instances are represented by numbered circles.

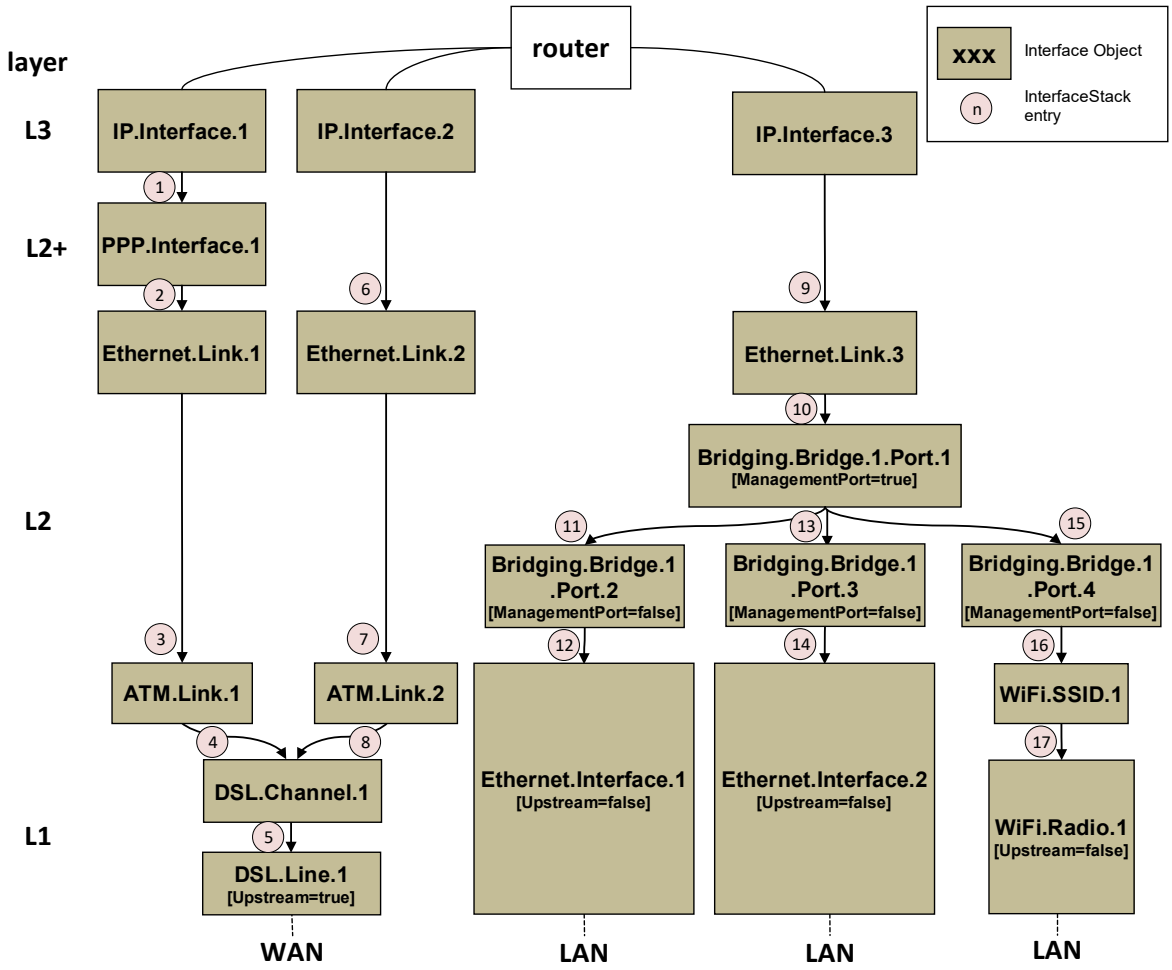



**Figure 14 – Simple Router Example (Interfaces Visualized)**

*NOTE – "Device." should be considered prepended to each parameter name in Figure 14. It is left off to make the figure more legible.*

Finally, Table 2 completes the example by listing each interface instance and its corresponding LowerLayers parameter value.

**Table 2 – Simple Router Example (Interface LowerLayers)**

| Interface | LowerLayers value |
|---|---|
| Device.IP.Interface.1 | Device.PPP.Interface.1 |
| Device.IP.Interface.2 | Device.Ethernet.Link.2 |
| Device.IP.Interface.3 | Device.Ethernet.Link.3 |
| Device.PPP.Interface.1 | Device.Ethernet.Link.1 |
| Device.Ethernet.Link.1 | Device.ATM.Link.1 |
| Device.Ethernet.Link.2 | Device.ATM.Link.2 |
| Device.Ethernet.Link.3 | Device.Bridging.Bridge.1.Port.1 |
| Device.Bridging.Bridge.1.Port.1 | Device.Bridging.Bridge.1.Port.2, Device.Bridging.Bridge.1.Port.3, Device.Bridging.Bridge.1.Port.4 |

| Interface | LowerLayers value |
|---|---|
| Device.Bridging.Bridge.1.Port.2 | Device.Ethernet.Interface.1 |
| Device.Bridging.Bridge.1.Port.3 | Device.Ethernet.Interface.2 |
| Device.Bridging.Bridge.1.Port.4 | Device.WiFi.SSID.1 |
| Device.ATM.Link.1 | Device.DSL.Channel.1 |
| Device.ATM.Link.2 | Device.DSL.Channel.1 |
| Device.DSL.Channel.1 | Device.DSL.Line.1 |
| Device.DSL.Line.1 | |
| Device.Ethernet.Interface.1 | |
| Device.Ethernet.Interface.2 | |
| Device.WiFi.SSID.1 | Device.WiFi.Radio.1 |
| Device.WiFi.Radio.1 | |

# 5 Parameter Definitions

The normative definition of the Device:2 data model is provided in XML DM Instance documents, as defined by TR-106 [3] Annex A.

For a given revision of the data model, the corresponding TR-181 Issue 2 XML document defines the Device:2 model itself and imports additional components from the other XML documents listed.

Each TR-181 Issue 2 HTML document is a report generated from the XML files, and lists a consolidated view of the Device:2 data model in human-readable form.

For use with CWMP the corresponding Device:2 data model is published at https://cwmp-data-models.broadband-forum.org, and for use with USP the data model is published at https://usp-data-models.broadband-forum.org.

# Annex A Bridging and Queuing

## A.1 Queuing and Bridging Model

Figure 15 shows the queuing and bridging model for a device. This model relates to the QoS object as well as the Bridging and Routing objects. The elements of this model are described in the following sections.

*NOTE – the queuing model described in this Annex is meant strictly as a model to clarify the intended behavior of the related data objects. There is no implication intended that an implementation has to be structured to conform to this model.*

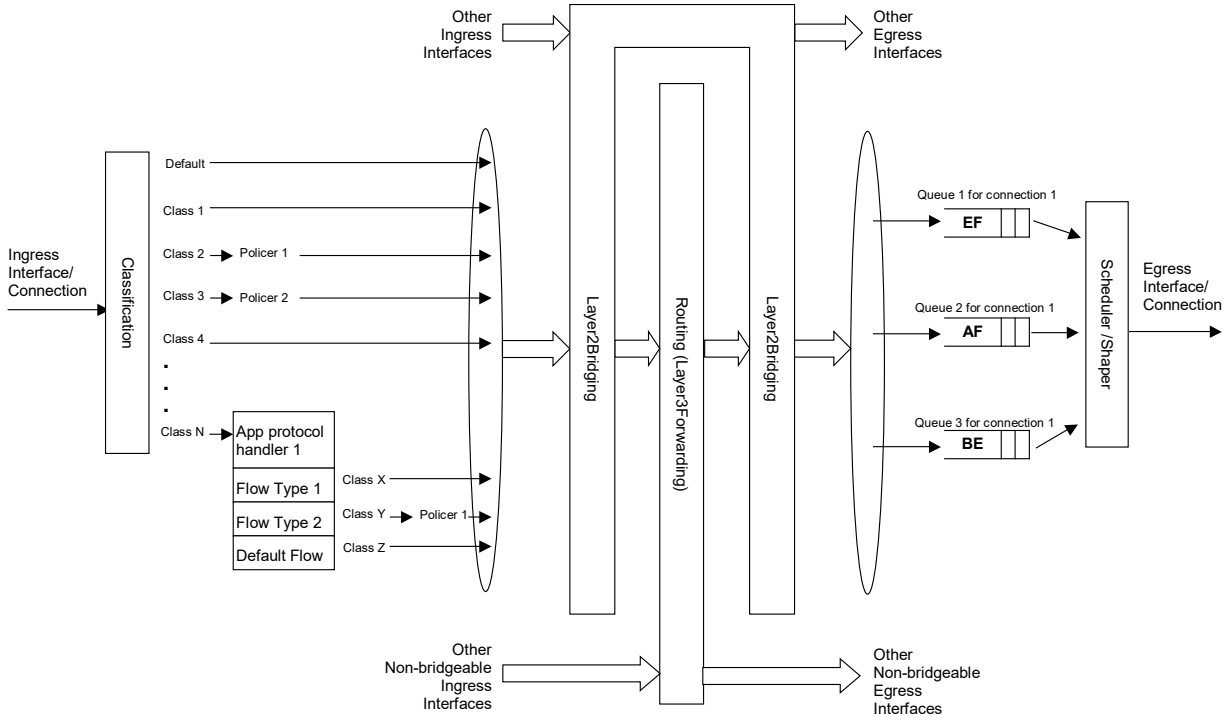



**Figure 15 – Queuing Model of a Device**

### A.1.1 Packet Classification

The Classification table within the QoS object specifies the assignment of each packet arriving at an ingress interface to a specific internal class. This classification can be based on a number of matching criteria, such as destination and source IP address, destination and source port, and protocol.

Each entry in the Classification table includes a series of parameters, each indicated to be a Classification Criterion. Each classification criterion can be set to a specified value, or can be set to a value that indicates that criterion is not to be used. A packet is defined to match the classification criteria for that table entry only if the packet matches all of the specified criteria. That is, a logical AND operation is applied across all classification criteria within a given Classification table entry.

*NOTE – to apply a logical OR to sets of classification criteria, multiple entries in the Classification table can be created that specify the same resulting queuing behavior.*

For each classification criterion, the Classification table also includes a corresponding "exclude" flag. This flag can be used to invert the sense of the associated classification criterion. That is, if this flag is *false* for a given criterion, the classifier is to include only packets that meet the specified criterion (as well as all others). If this flag is *true* for a given criterion, the classifier is to include all packets except those that meet the associated criterion (in addition to meeting all other criteria).

For a given entry in the Classification table, the classification is to apply only to the interface specified by the Interface parameter. This parameter can specify a particular ingress interface or all sources. Depending on the particular interface, not all classification criteria will be applicable. For example, Ethernet layer classification criteria would not apply to packets arriving on a non-bridged ATM VC.

Packet classification is modeled to include all ingress packets regardless of whether they ultimately will be bridged or routed through the device.

#### A.1.1.1 Classification Order

The class assigned to a given packet corresponds to the first entry in the Classification table (given the specified order of the entries in the table) whose matching criteria match the packet. If there is no entry that matches the packet, the packet is assigned to a default class.

Classification rules are sensitive to the order in which they are applied because certain traffic might meet the criteria of more than one Classification table entry. The Order parameter is responsible for identifying the order in which the Classification entries are to be applied.

The following rules apply to the use and setting of the Order parameter:

- Order goes in order from 1 to n, where n is equal to the number of entries in the Classification table. 1 is the highest precedence, and n the lowest. For example, if entries with Order of 4 and 7 both have rules that match some particular traffic, the traffic will be classified according to the entry with the 4.
- The CPE is responsible for ensuring that all Order values are unique and sequential.
  - If an entry is added (number of entries becomes n+1), and the value specified for Order is greater than n+1, then the CPE will set Order to n+1.
  - If an entry is added (number of entries becomes n+1), and the value specified for Order is less than n+1, then the CPE will create the entry with that specified value, and increment the Order value of all existing entries with Order equal to or greater than the specified value.
  - If an entry is deleted, the CPE will decrement the Order value of all remaining entries with Order greater than the value of the deleted entry.
  - If the Order value of an entry is changed, then the value will also be changed for other entries greater than or equal to the lower of the old and new values, and less than the larger of the old and new values. If the new value is less than the old, then these other entries will all have Order incremented. If the new value is greater than the old, then the other entries will have Order decremented and the changed entry will be given a value of

<new value>-1. For example, an entry is changed from 8 to 5. The existing 5 goes to 6, 6 to 7, and 7 to 8. If the entry goes from 5 to 8, then 6 goes to 5, 7 to 6, and the changed entry is 7. This is consistent with the behavior that would occur if the change were considered to be an Add of a new entry with the new value, followed by a Delete of the entry with the old value.

#### A.1.1.2 Dynamic Application Specific Classification

In some situations, traffic to be classified cannot be identified by a static set of classification criteria. Instead, identification of traffic flows might require explicit application awareness. The model accommodates such situations via the App and Flow tables in the QoS object.

Each entry in the App table is associated with an application-specific protocol handler, identified by the ProtocolIdentifier, which contains a URN. For a particular CPE, the AvailableAppList parameter indicates which protocol handlers that CPE is capable of supporting, if any. A list of standard protocol handlers and their associated URNs is specified in Section A.3, though a CPE can also support vendor-specific protocol handlers as well. Multiple App table entries can refer to the same ProtocolIdentifier.

The role of the protocol handler is to identify and classify flows based on application awareness. For example, a SIP protocol handler might identify a call-control flow, an audio flow, and a video flow. The App and Flow tables are used to specify the classification outcome associated with each such flow.

For each App table entry there can be one or more associated Flow table entries. Each flow table entry identifies a type of flow associated with the protocol handler. The Type parameter is used to identify the specific type of flow associated with each entry. For example, a Flow table entry for a SIP protocol handler might refer only to the audio flows associated with that protocol handler. A list of standard flow type values is given in Section A.3, though a CPE can also support vendor-specific flow types.

A protocol handler can be defined as being fed from the output of a Classification table entry. That is, a Classification entry can be used to single out control traffic to be passed to the protocol handler, which then subsequently identifies associated flows. Doing so allows more than one instance of a protocol handler associated with distinct traffic. For example, one could define two App table entries associated with SIP protocol handlers. If the classifier distinguished control traffic to feed into each handler based on the destination IP address of the SIP server, this could be used to separately classify traffic for different SIP service providers. In this case, each instance of the protocol handler would identify only those flows associated with a given service. Note that the Classification table entry that feeds each protocol handler wouldn't encompass all of the flows; only the traffic needed by the protocol handler to determine the flows—typically only the control traffic.

#### A.1.1.3 Classification Outcome

Each Classification entry specifies a tuple composed of either:
A TrafficClass and (optionally) a Policer, or
An App table entry

Each entry also specifies:
Outgoing DiffServ and Ethernet priority marking behavior
A ForwardingPolicy tag that can be referenced in the Routing table to affect packet routing (note that the ForwardingPolicy tag affects only routed traffic)
Note that the information associated with the classification outcome is modeled as being carried along with each packet as it flows through the system.

If a packet does not match any Classification table entry, the DefaultTrafficClass, DefaultPolicer, default markings, and default ForwardingPolicy are used.

If a TrafficClass/Policer tuple is specified, classification is complete. If, however, an App is specified, the packet is passed to the protocol handler specified by the ProtocolIdentifier in the specified App table entry for additional classification (see Section A.1.1.2). If any of the identified flows match the Type specified in any Flow table entry corresponding to the given App table entry (this correspondence is indicated by the App identifier), the specified tuple and markings for that Flow table entry is used for packets in that flow. Other flows associated with the application, but not explicitly identified, use the default tuple and markings specified for that App table entry.

### A.1.2 Policing

The Policer table defines the policing parameters for ingress packets identified by either a Classification table entry (or the default classification) or a dynamic flow identified by a protocol handler identified in the App table.

Each Policer table entry specifies the packet handling characteristics, including the rate requirements and behavior when these requirements are exceeded.

### A.1.3 Queuing and Scheduling

The Queue table specifies the number and types of queues, queue parameters, shaping behavior, and scheduling algorithm to use. Each Queue table entry specifies the TrafficClasses with which it is associated, and a set of egress interfaces for which a queue with the corresponding characteristics needs to exist.

*NOTE – If the CPE can determine that among the interfaces specified for a queue to exist, packets classified into that queue cannot egress to a subset of those interfaces (from knowledge of the current routing and bridging configuration), the CPE can choose not to instantiate the queue on those interfaces.*

*NOTE – Packets classified into a queue that exit through an interface for which the queue is not specified to exist, will instead use the default queuing behavior. The default queue itself will exist on all egress interfaces.*

The model defined here is not intended to restrict where the queuing is implemented in an actual implementation. In particular, it is up to the particular implementation to determine at what protocol layer it is most appropriate to implement the queuing behavior (IP layer, Ethernet MAC layer, ATM layer, etc.). In some cases, however, the QoS configuration would restrict the choice of layer where queuing can be implemented. For example, if a queue is specified to carry traffic that is bridged, then it could not be implemented as an IP-layer queue.

*NOTE – care needs to be taken to avoid having multiple priority queues multiplexed onto a single connection that is rate shaped. In such cases, the possibility exists that high priority traffic can be held back due to rate limits of the overall connection exceeded by lower priority traffic. Where possible, each priority queue will be shaped independently using the shaping parameters in the Queue and Shaping table.*

The scheduling parameters defined in the Queue table apply to the first level of what might be a more general scheduling hierarchy. This specification does not specify the rules that an implementation needs to apply to determine the most appropriate scheduling hierarchy given the scheduling parameters defined in the Queue table.

As an example, take a situation where the output of four distinct queues is to be multiplexed into a single connection, and two entries share one set of scheduling parameters while the other two entries share a different set of scheduling parameters. In this case, it might be appropriate to implement this as a scheduling hierarchy with the first two queues multiplexed with a scheduler defined by the first pair, and the second two queues being multiplexed with a scheduler defined by the second pair. The lower layers of this scheduling hierarchy cannot be directly determined from the content of the Queue table.

### A.1.4 Bridging

*NOTE – from the point of view of a bridge, packets arriving into the bridge from the local router (either upstream or downstream) are treated as ingress packets, even though the same packets, which just left the router, are treated as egress from the point of view of the router. For example, a Filter table entry might admit packets on ingress to the bridge from a particular IP interface, which means that it admits packets on their way out of the router over this layer 3 connection.*

For each interface, the output of the classifier is modeled to feed a set of 802.1D [8] or 802.1Q [9] layer 2 bridges as specified by the Bridging object. Each bridge specifies layer 2 connectivity between one or more layer 2 downstream and/or upstream interfaces, and optionally one or more layer 3 connections to the local router.

Each bridge corresponds to a single entry in the Bridge table of the Bridging object. The Bridge table contains the following sub-tables:
Port table: models the Bridge ports, which are either management ports (modeling layer 3 connections to the local router) or non-management ports (modeling connections to layer 2 interfaces). Bridge ports are stackable interface objects (see Section 4.2).
VLAN table: models the Bridge VLANs (relevant only to 802.1Q bridges).
VLANPort table: for each VLAN, defines the ports that comprise its member set (relevant only to 802.1Q bridges).

#### A.1.4.1 Filtering

Traffic from a given interface (or set of interfaces) can be selectively admitted to a given Bridge, rather than bridging all traffic from that interface. Each entry in the Filter table includes a series of classification criteria. Each classification criterion can be set to a specified value, or can be set to a value that indicates that criterion is not to be used. A packet is admitted to the Bridge only if

the packet matches all of the specified criteria. That is, a logical AND operation is applied across all classification criteria within a given Filter table entry.

*NOTE – to apply a logical OR to sets of classification criteria, multiple entries in the Filter table can be created that refer to the same interfaces and the same Bridge table entry.*

*NOTE – a consequence of the above rule is that, if a packet does not match the criteria of any of the enabled Filter table entries, then it will not be admitted to any bridges, i.e. it will be dropped. As a specific example of this, if none of the enabled Filter table entries reference a given interface, then all packets arriving on that interface will be dropped.*

For each classification criterion, the Filter table also includes a corresponding "exclude" flag. This flag can be used to invert the sense of the associated classification criterion. That is, if this flag is false for a given criterion, the Bridge will admit only packets that meet the specified criterion (as well as all other criteria). If this flag is true for a given criterion, the Bridge will admit all packets except those that meet the associated criterion (in addition to meeting all other criteria).

Note that because the classification criteria are based on layer 2 packet information, if the selected port for a given Filter table entry is a layer 3 connection from the local router, the layer 2 classification criteria do not apply.

### A.1.4.2 Filter Order

Any packet that matches the filter criteria of one or more filters is admitted to the Bridge associated with the first entry in the Filter table (relative to the specified Order).

The following rules apply to the use and setting of the Order parameter:
The Order goes in order from 1 to n, where n is equal to the number of filters. 1 is the highest precedence, and n the lowest.
The CPE is responsible for ensuring that all Order values among filters are unique and sequential.
If a filter is added (number of filters becomes n+1), and the value specified for Order is greater than n+1, then the CPE will set Order to n+1.
If a filter is added (number of entries becomes n+1, and the value specified for Order is less than n+1, then the CPE will create the entry with that specified value, and increment the Order value of all existing filters with Order equal to or greater than the specified value.
If a filter is deleted, the CPE will decrement the Order value of all remaining filters with Order greater than the value of the deleted entry.
If the Order value of a filter is changed, then the value will also be changed for other filters greater than or equal to the lower of the old and new values, and less than the larger of the old and new values. If the new value is less than the old, then these other entries will all have Order incremented. If the new value is greater than the old, then the other entries will have Order decremented and the changed entry will be given a value of <new value>-1. For example, an entry is changed from 8 to 5. The existing 5 goes to 6, 6 to 7, and 7 to 8. If the entry goes from 5 to 8, then 6 goes to 5, 7 to 6, and the changed entry is 7. This is consistent with the behavior that would occur if the change were considered to be an Add of a new filter with the new value, followed by a Delete of the filter with the old value.

## A.2 Default Layer 2/3 QoS Mapping

Table 3 presents a "default" mapping between layer 2 and layer 3 QoS. In practice, it is a guideline for automatic marking of DSCP (layer 3) based upon Ethernet Priority (layer 2) and the other way around. Please refer to the QoS Classification table's DSCPMark and EthernetPriorityMark parameters (and related parameters) for configuration of a default automatic DSCP / Ethernet Priority mapping.

Automatic marking of DSCP or Ethernet Priority is likely only in the following cases:
WAN → LAN: to map DSCP (layer 3) to Ethernet Priority (layer 2)
LAN → WAN: to map Ethernet Priority (layer 2) to DSCP (layer 3)
Automatic marking in the LAN → LAN case is unlikely, since LAN QoS is likely to be supported only at layer 2, and LAN DSCP values, if used, will probably be a direct representation of Ethernet Priority, e.g. Ethernet Priority shifted left by three bits.

In the table, grayed and bolded items are added to allow two-way mapping between layer 2 and layer 3 QoS (where the mapping is ambiguous, the grayed values SHOULD be ignored and the bolded values SHOULD be used). If, when mapping from layer 3 to layer 2 QoS, the DSCP value is not present in the table, the mapping SHOULD be based only on the first three bits of the DSCP value, i.e. on DSCP & 111000.

**Table 3 – Default Layer 2/3 QoS Mapping**

| Layer 2 | | Layer 3 | |
|---|---|---|---|
| Ethernet Priority | Designation | DSCP | Per Hop Behavior |
| 001 (1) | BK | 000000 (0x00) | Default |
| 010 (2) | spare | 000000 (0x00) | |
| 000 (0) | BE | 000000 (0x00)<br>**000000 (0x00)** | Default<br>CS0 |
| 011 (3) | EE | 001110 (0x0e)<br>001100 (0x0c)<br>001010 (0x0a)<br>**001000 (0x08)** | AF13<br>AF12<br>AF11<br>CS1 |
| 100 (4) | CL | 010110 (0x16)<br>010100 (0x14)<br>010010 (0x12)<br>**010000 (0x10)** | AF23<br>AF22<br>AF21<br>CS2 |
| 101 (5) | VI | 011110 (0x1e)<br>011100 (0x1c)<br>011010 (0x1a)<br>**011000 (0x18)** | AF33<br>AF32<br>AF31<br>CS3 |
| 110 (6) | VO | 100110 (0x26)<br>100100 (0x24)<br>100010 (0x22)<br>**100000 (0x20)** | AF43<br>AF42<br>AF41<br>CS4 |
| 110 (6) | VO | 101110 (0x2e)<br>**101000 (0x28)** | EF<br>CS5 |
| 111 (7) | NC | 110000 (0x30)<br>**111000 (0x38)** | CS6<br>CS7 |

## A.3 URN Definitions for App and Flow Tables

### A.3.1 App ProtocolIdentifier

Table 4 lists the URNs defined for the QoS App table's ProtocolIdentifier parameter. Additional standard or vendor-specific URNs can be defined following the standard syntax for forming URNs.

**Table 4 – ProtocolIdentifer URNs**

| URN | Description |
|---|---|
| urn:dslforum-org:sip | Session Initiation Protocol (SIP) as defined by RFC 3261 [12] |
| urn:dslforum-org:h.323 | ITU-T Recommendation H.323 |
| urn:dslforum-org:h.248 | ITU-T Recommendation H.248 (MEGACO) |
| urn:dslforum-org:mgcp | Media Gateway Control Protocol (MGCP) as defined by RFC 3435 [13] |
| urn:dslforum-org:pppoe | Bridged sessions of PPPoE |

### A.3.2 Flow Type

A syntax for forming URNs for the QoS Flow table's Type parameter is defined for the Session Description Protocol (SDP) as defined by RFC 4566 [14]. Additional standard or vendor-specific URNs can be defined following the standard syntax for forming URNs.

A URN to specify an SDP flow is formed as follows:
urn:dslforum-org:sdp-[MediaType]-[Transport]

[MediaType] corresponds to the "media" sub-field of the "m" field of an SDP session description.
[Transport] corresponds to the "transport" sub-field of the "m" field of an SDP session description.
Non-alphanumeric characters in either field are removed (e.g., "rtp/avp" becomes "rtpavp").

For example, the following would be valid URNs referring to SDP flows:
urn:dslforum-org:sdp-audio-rtpavp
urn:dslforum-org:sdp-video-rtpavp
urn:dslforum-org:sdp-data-udp

For flow type URNs following this convention, there is no defined use for TypeParameters, which SHOULD be left empty.

For the ProtocolIdentifier urn:dslforum-org:pppoe, a single flow type is defined referring to the entire PPPoE session. The URL for this flow type is:
urn:dslforum-org:pppoe

### A.3.3 Flow TypeParameters

For the flow type urn:dslforum-org:pppoe, Table 5 specifies the defined TypeParameter values.

**Table 5 – Flow TypeParameters values for flow type urn:dslforum-org:pppoe**

| Name | Description of Value |
|---|---|
| ServiceName | The PPPoE service name.<br>If specified, only bridged PPPoE sessions designated for the named service would be considered part of this flow.<br>If this parameter is not specified, or is empty, bridged PPPoE associated with any service considered part of this flow. |
| ACName | The PPPoE access concentrator name.<br>If specified, only bridged PPPoE sessions designated for the named access concentrator would be considered part of this flow.<br>If this parameter is not specified, or is empty, bridged PPPoE associated with any access concentrator considered part of this flow. |
| PPPDomain | The domain part of the PPP username.<br>If specified, only bridged PPPoE sessions in which the domain portion of the PPP username matches this value are considered part of this flow.<br>If this parameter is not specified, or is empty, all bridged PPPoE sessions are considered part of this flow. |

# Annex B Tunneling

## B.1 Overview

Consider a device that provides a layer 3 tunnel endpoint. Some packets will need to be en-tunneled and then will leave the device in the tunnel. Other packets will arrive at the device in the tunnel and will need to be de-tunneled. This is illustrated in Figure 16, in which green indicates application traffic, yellow indicates an IP interface, and pink indicates a tunnel (carrying green application traffic).

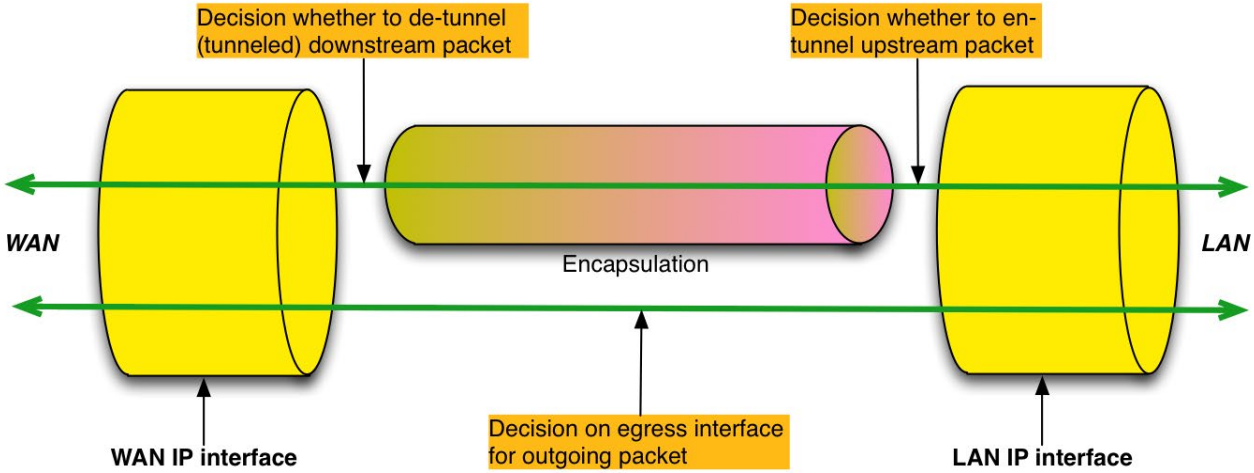



**Figure 16 – Tunneling Overview**

The Figure highlights three decisions:

1. Whether to en-tunnel an upstream packet.
2. Whether to de-tunnel a downstream packet.
3. To which egress interface to send an outgoing packet.

This egress interface decision is just a normal forwarding decision. By separately modeling the *Tunnel interface* and the *Tunnel*, the Device:2 data model is able to present the en-tunnel decision as also being a forwarding decision. The de-tunnel decision is not really a decision at all, because it happens automatically as a result of normal packet processing.

This modeling approach imposes no restrictions on the device implementation; it is just how the en-tunnel and de-tunnel decisions are modeled.

- Each *Tunnel* instance models a tunnel and has one or more *Tunnel interface* children, each of which models a flow / session within that tunnel. These *Tunnel interface* children are stackable interface objects.

- Upstream traffic that is to be en-tunneled is routed to a *Tunnel interface* instance, is passed to the parent *Tunnel* instance, is encapsulated, and then arrives on the *Tunnel* instance.

- Downstream traffic that is to be de-tunneled is passed to a *Tunnel* instance, is de-encapsulated, and then arrives on the appropriate child *Tunnel interface* instance.

- Traffic arriving on a *Tunnel* or on a *Tunnel interface* is classified, marked, policed, bridged, routed and queued in the same way as traffic arriving on any other interface.

*NOTE – a Tunnel is not a stackable interface object, because it breaks the layering order and can be regarded as separating two different protocol stacks, one of which acts as a carrier for the other. This is clearly illustrated in Figure 20 and the other interface stack Figures.*

*NOTE – even though a Tunnel is not an interface, it can be referenced by QoS classification rules. Traffic arriving on a Tunnel instance, i.e. packets that have just been encapsulated, is conceptually similar to locally-generated traffic.*

In summary, the decision to en-tunnel a packet is a forwarding decision to send a packet to an IP interface that is stacked above a *Tunnel interface* instance, and the decision to de-tunnel a packet is a consequence of the fact that it is addressed to the CPE and is therefore passed to a *Tunnel* instance. Figure 17 extends Figure 16 by expanding the tunnel into a *Tunnel IP interface*, a *Tunnel interface*, and the *Tunnel* instance, thereby showing where these two decisions are made.

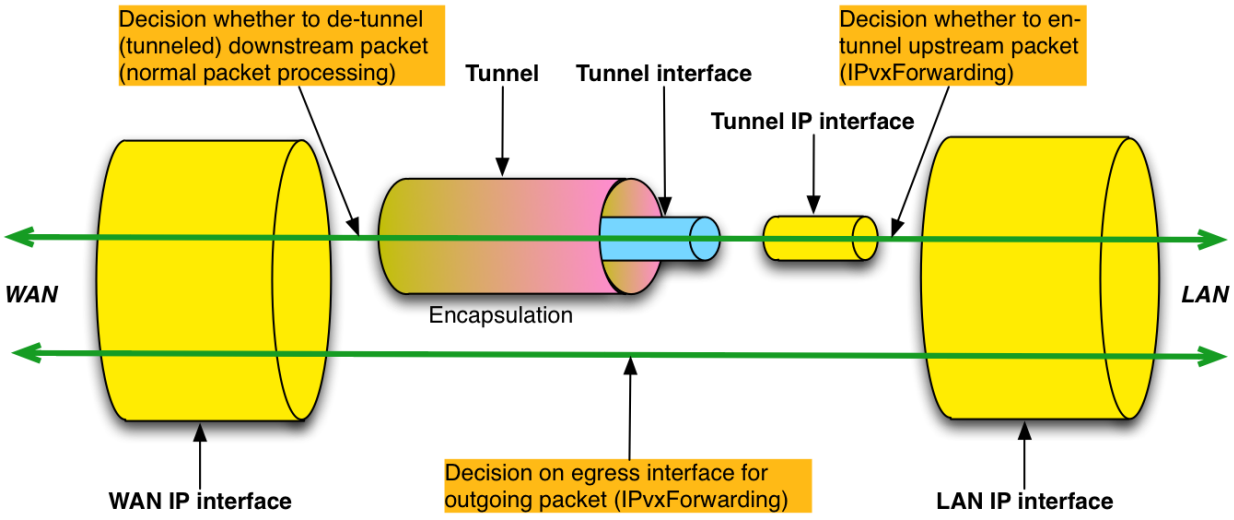



**Figure 17 – Tunneling Overview (Showing Forwarding Decisions)**

*NOTE – the existing 6rd, DS-Lite and IPsec data models use a less flexible approach in which the Tunnel interfaces are not explicitly modeled, and a separate non-stackable Tunnel table references auto-created Tunnel/Tunneled IP interface pairs. See B.2 for further details.*

*NOTE – the Tunnel interface and Tunnel approach is more flexible because (a) it supports multiple flows / sessions with a tunnel (e.g. GRE traffic flows or L2TP sessions), (b) it supports additional encapsulation layers between the Tunnel IP interface and the Tunnel interface (e.g. PPP for L2TP), and (c) it supports layer 2 tunneling use cases (traffic is bridged directly to the Tunnel interface and there is no Tunnel IP interface). See B.2 for further details.*

Figure 18 and Figure 19 show upstream and downstream examples of how the *Tunnel interface* and *Tunnel* instances are used to describe the traffic path through the device for both untunneled and tunneled packets.

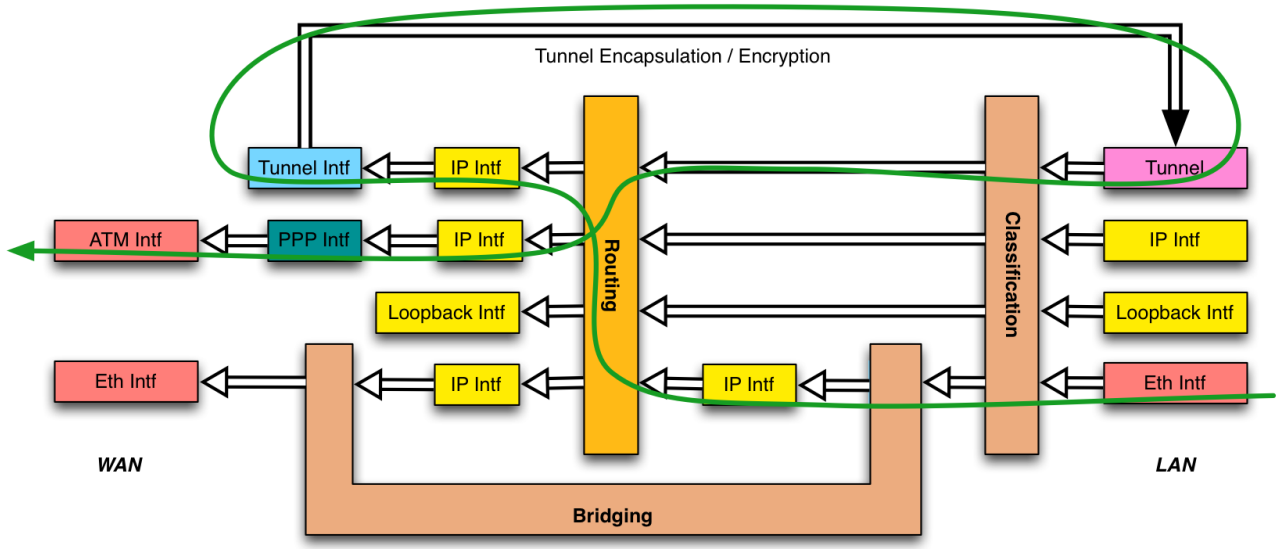



**Figure 18 – Sample Flow of Upstream Tunneled Traffic through the Device**

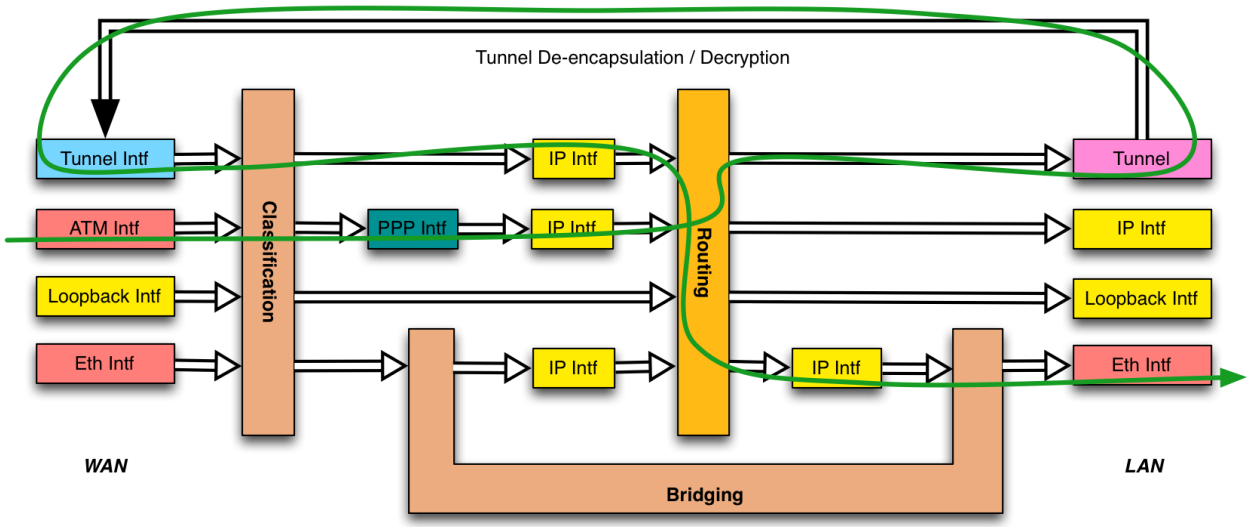



**Figure 19 – Sample Flow of Downstream Tunneled Traffic through the Device**

The less flexible (*Tunnel,Tunneled*) IP interface mechanism is used in the following three cases:

- IPv6rd (Appendix VI) *Device.IPv6rd*.
- DS-Lite (Appendix VII) *Device.DSLite*.
- IPsec (Appendix IX) *Device.IPsec*.

The flexible *Tunnel interface* and *Tunnel* mechanism is used for the following two cases and will be used for modeling all future tunneling scenarios:

- GRE (Appendix XIV) *Device.GRE*.
- MAP (Appendix XV) *Device.MAP*.

## B.2 Details

Figure 20 shows the interface stack for a general layer 3 tunneling scenario. Compare with Figure 21, which is derived from Figure 17. It can be seen that each Figure presents a different view of the same thing.

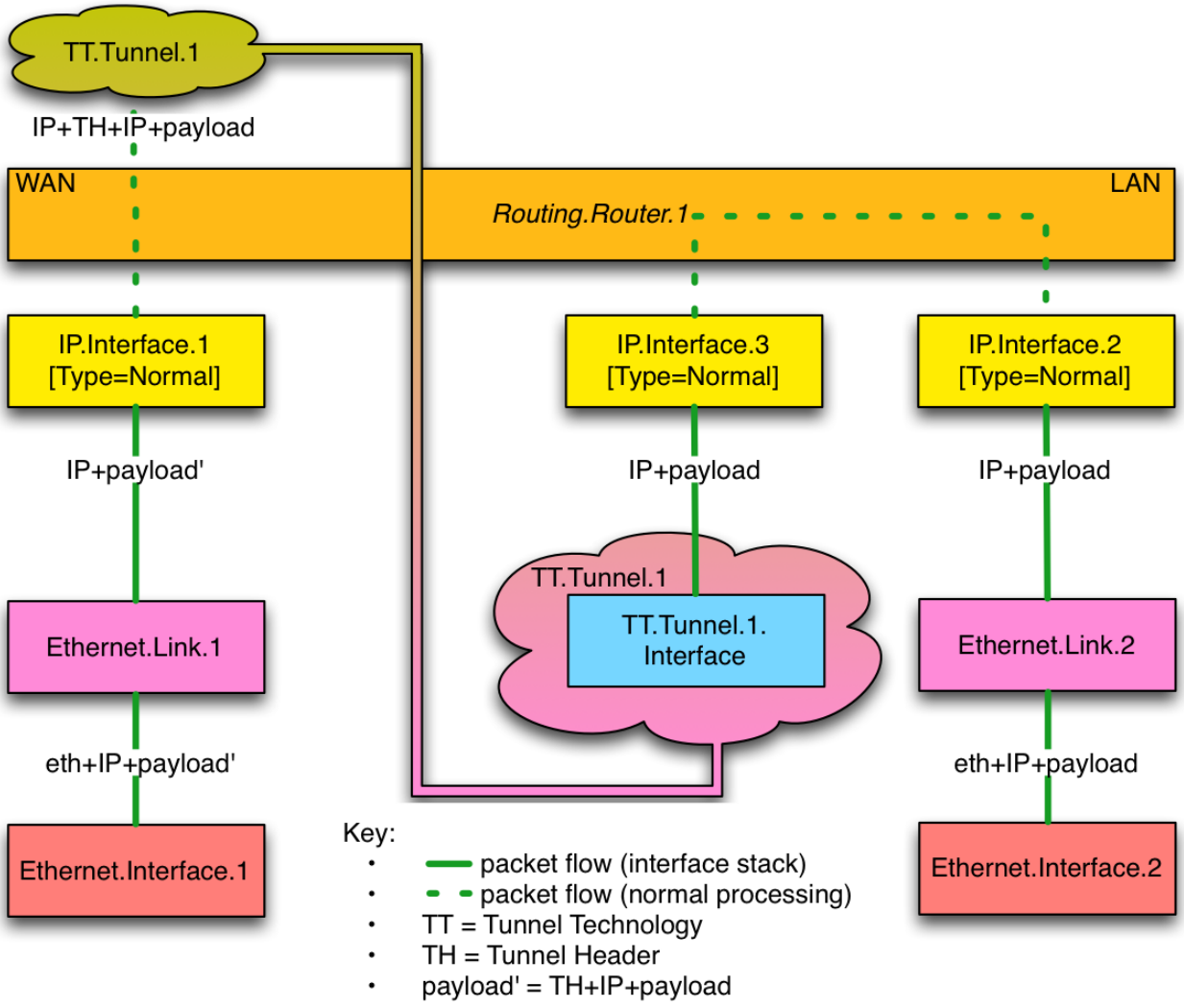



**Figure 20 – General Layer 3 Tunneling Interface Stack**

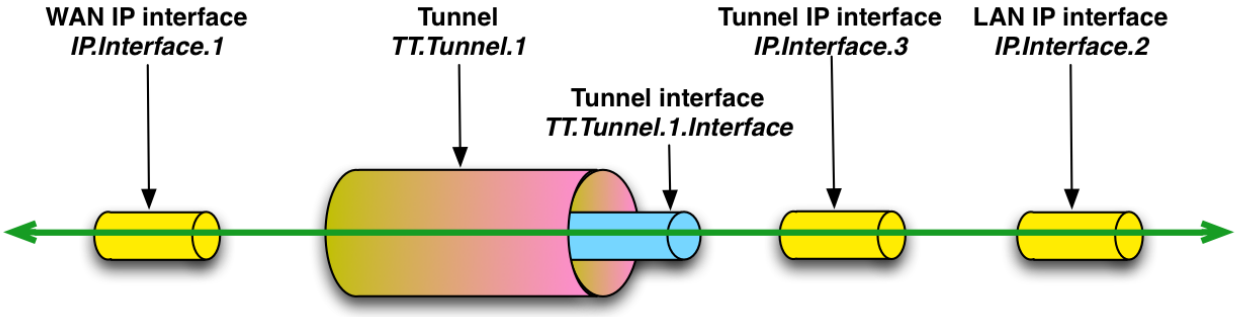



**Figure 21 – General Layer 3 Tunneling (from Figure 17)**

*NOTE – IP.Interface.3 is labeled as Type=Normal in Figure 20 but as Tunnel IP interface in Figure 21. IP interface Type=Tunnel was defined specifically for the (Tunnel,Tunneled) IP interface mechanism and is not needed because IP.Interface.3 is stacked above TT.Tunnel.1.Interface.1.*

*Figure 20 is general in that it is independent of the tunnel technology, but it doesn't illustrate all the possibilities. If supported by the tunnel technology:*

- *A Tunnel can have multiple Tunnel interface children, each of which models a flow or session. In this case the Tunnel interface object is multi-instance.*
- *There can be additional encapsulation layers between the Tunnel IP interface(s) and the Tunnel interface(s).*

*Figure 22 shows an L2TP [44] example that illustrates both of the above.*

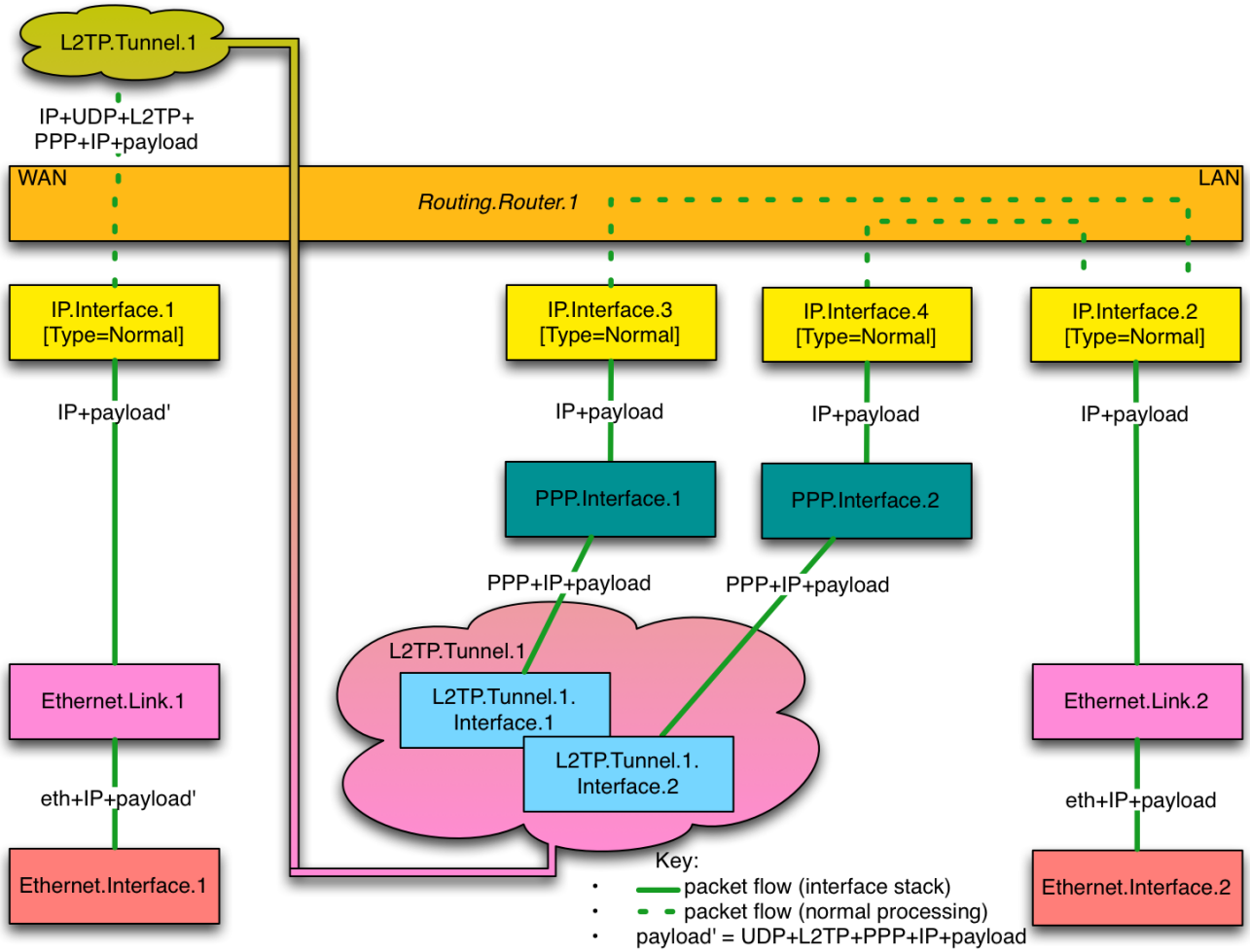



**Figure 22 – L2TP Interface Stack Example**

Some tunneling technologies support layer 2 tunnels, in which the tunnel payload is a layer 2 packet. Figure 23 shows the interface stack for a general layer 2 tunneling scenario. This is conceptually similar to the layer 3 case, but a bridge port rather than an IP interface is stacked above the *Tunnel interface*.

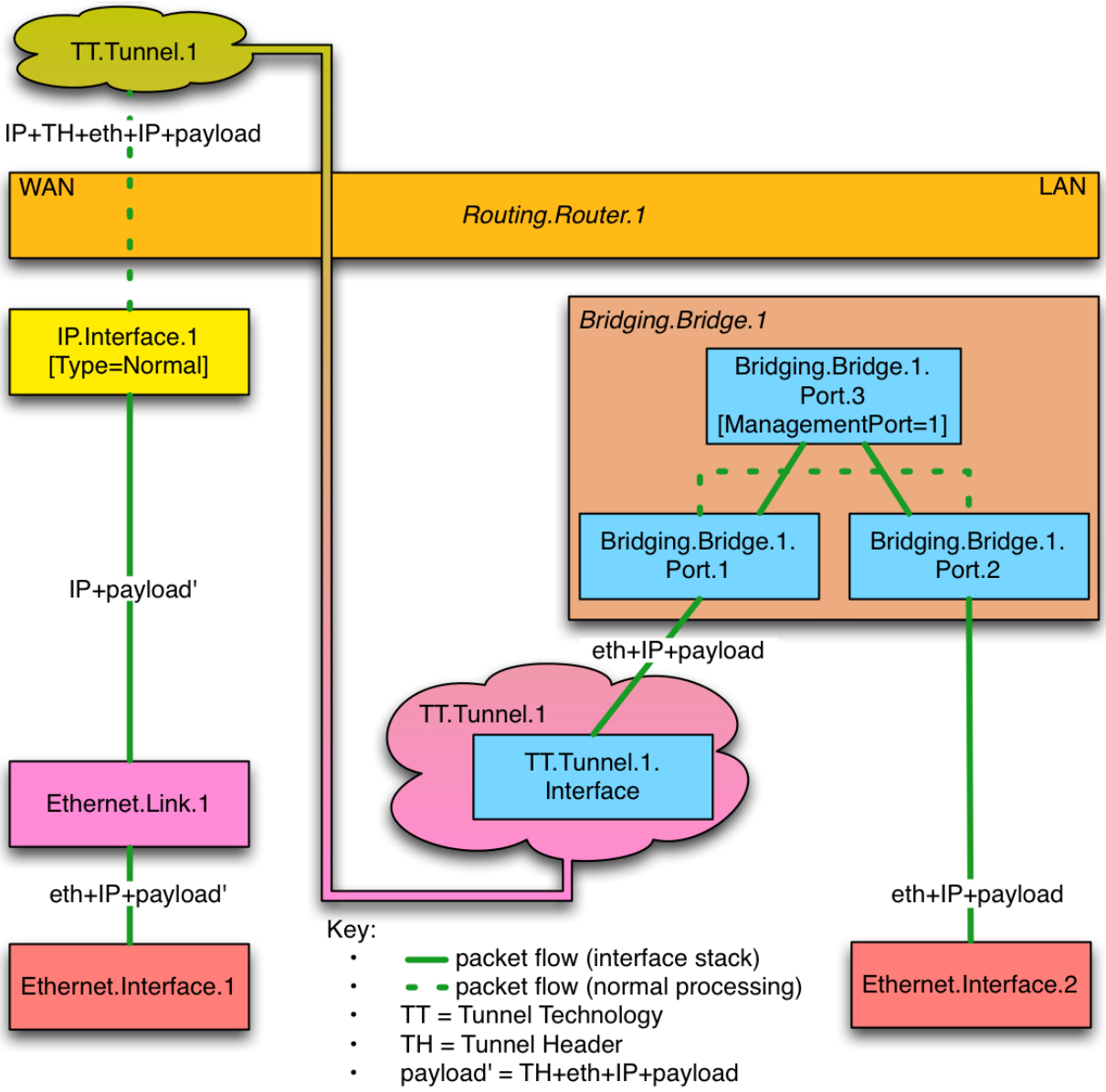



**Figure 23 – General Layer 2 Tunneling Interface Stack**

# Annex C Software Module Management UUID Usage

## C.1 Overview

The Software Module Management mechanism uses a UUID (see RFC 4122 [66] for a complete definition of UUID) to uniformly identify a Deployment Unit across Agents. Since Deployment Units can be installed multiple times on a single Agent (e.g. multiple versions of the same Deployment Unit or the same version of the Deployment Unit on different Execution Environments), a Deployment Unit on a specific Agent is uniquely identified by the combination of UUID, version, and Execution Environment that the Deployment Unit is installed upon, but the UUID is still the uniform unique identifier of that Deployment Unit (i.e. this means that the UUID will be the same independent of the version of the Deployment Unit). A version 5 UUID is a method for generating UUIDs from "names" that are unique within some "namespace", which means that a UUID generated by different actors but using the same "name" and "namespace" will cause the generation of the same exact UUID. The Software Module Management mechanism requires, whether the Controller or the Agent generates the UUID, that the UUID be generated in the exact same manner following both the rules defined in Section 4.3 / RFC 4122 [66] and the rules defined within this Annex.

Section 4.3 / RFC 4122 [66] identifies the following high-level requirements for a Version 3 UUID:

- The UUIDs generated at different times from the same name in the same namespace MUST be equal.
- The UUIDs generated from two different names in the same namespace should be different (with very high probability).
- The UUIDs generated from the same name in two different namespaces should be different with (very high probability).
- If two UUIDs that were generated from names are equal, then they were generated from the same name in the same namespace (with very high probability).

The remainder of this Annex defines additional rules that MUST be followed by the Controller and Agent when generating a UUID as well as under what circumstances a Agent will be required to generate a UUID.

## C.2 UUID Generation Requirements

The following set of additional requirements ensures that the Version 5 UUID will be uniform regardless of whether the Controller or Agent generates it:

The FQDN "namespace" UUID as defined in Appendix C /RFC 4122 [66] MUST be used: 6ba7b810-9dad-11d1-80b4-00c04fd430c8

The "name" will be the FQDN of the Deployment Unit, which MUST be a combination of the Deployment Unit's Name (the value that will be contained within the DeploymentUnit.{i}.Name Parameter) and the Deployment Unit Vendor's domain name (the value that will be contained within the DeploymentUnit.{i}.Vendor Parameter). The format is: *'<Name> + "." + <Vendor> + "."'*. For example, if the DU Vendor is "broadband-forum.org" and the DU Name is "sample1", then the FQDN of the DU is "sample1.broadband-forum.org."

*NOTE - As the Deployment Unit's Name is used within generation of the FQDN, it MUST be altered if it contains any characters other than 0-9, a-z, A-Z, _ (underscore), or – (hyphen). Percent-*

*encoding MUST be used to replace any other characters (i.e. a '%' character followed by the ASCII hex value of the replaced character). For example, a Deployment Unit Name of "sample.1" would be converted to "sample%2e1".*

An example of a Version 5 UUID looks like:
76183ed7-6a38-3890-66ef-a6488efb6690

## C.3 Agent Requirements

There are three circumstances when a Agent MUST generate its own UUID:
Factory-Installed Deployment Units : a Deployment Unit is installed at factory time without the aid of a Controller
LAN-Side-Installed Deployment Units : a Deployment Unit is installed by a LAN-Side management mechanism (e.g. UPnP DM SMS, CLI, or GUI) without the aid of a Controller
Controller-Installed Deployment Units : a Deployment Unit is installed by a Controller, but the Controller either does not send the UUID or sends an empty string as the UUID within the Install operation of the ChangeDUState RPC.
In these circumstances the Agent MUST generate the UUID as it installs the Deployment Unit. The Controller can discover / validate the generated UUID by either inspecting the DUStateChangeComplete or inspecting the Deployment Unit Data Model table.

# Appendix I Example RG Queuing Architecture

The queuing and scheduling discipline envisioned upstream for the RG is shown in Figure 24, taken from the description of TR-059 [55].

There are multiple access sessions supported in this model, however, all traffic is classified and scheduled in a monolithic system. So, while it might appear at first that the Diffserv queuing and scheduling might apply only to IP-aware access – in fact all access, IP, Ethernet, or PPP is managed by the same system that adheres to the Diffserv model.

For example, at the bottom of the figure, BE treatment is given to the non-IP-aware access sessions (PPPoE started behind the RG or delivered to an L2TP tunnel delivery model). This queue might be repeated several times in order to support fairness among multiple PPPoE accesses – or it can be a monolithic queue with separate rate limiters applied to the various access sessions.

The PTA access is a single block of queues. This is done because NSP access typically works with a single default route to the NSP, and managing more than one simultaneously at the RG would be perilous. The $\sum$ rate limiter would limit the overall access traffic for a service provider.

Rate limiters are also shown within the EF and AF service classes because the definition of those Diffserv types is based on treating the traffic differently when it falls into various rates.

Finally, at the top of the diagram is the ASP access block of queues. In phase 1A, these queues are provisioned and provide aggregate treatment of traffic mapped to them. In phase 1B, it will become possible to assign AF queues to applications to give them specific treatment instead of aggregate treatment. The EF service class can also require a high degree of coordination among the applications that make use of it so that its maximum value is not exceeded.

Notable in this architecture is that all the outputs of the EF, AF, and BE queues are sent to a scheduler (**S**) that pulls traffic from them in a strict priority fashion. In this configuration EF traffic is, obviously, given highest precedence and BE is given the lowest. The AF service classes fall in-between.

Note that there is significant interest in being able to provide a service arrangement that would allow general Internet access to have priority over other (bulk rate) services.[5] Such an arrangement would be accomplished by assigning the bulk rate service class to BE and by assigning the default service class (Internet access) as AF with little or no committed information rate.

Given this arrangement, the precedence of traffic shown in the figure is arranged as:

a. EF – red dotted line

[5] *This "bulk rate" service class would typically be used for background downloads and potentially for peer-to-peer applications as an alternative to blocking them entirely.*

b. AF – blue dashed line (with various precedence among AF classes as described in RFC 2597 [10])
c. BE – black solid line

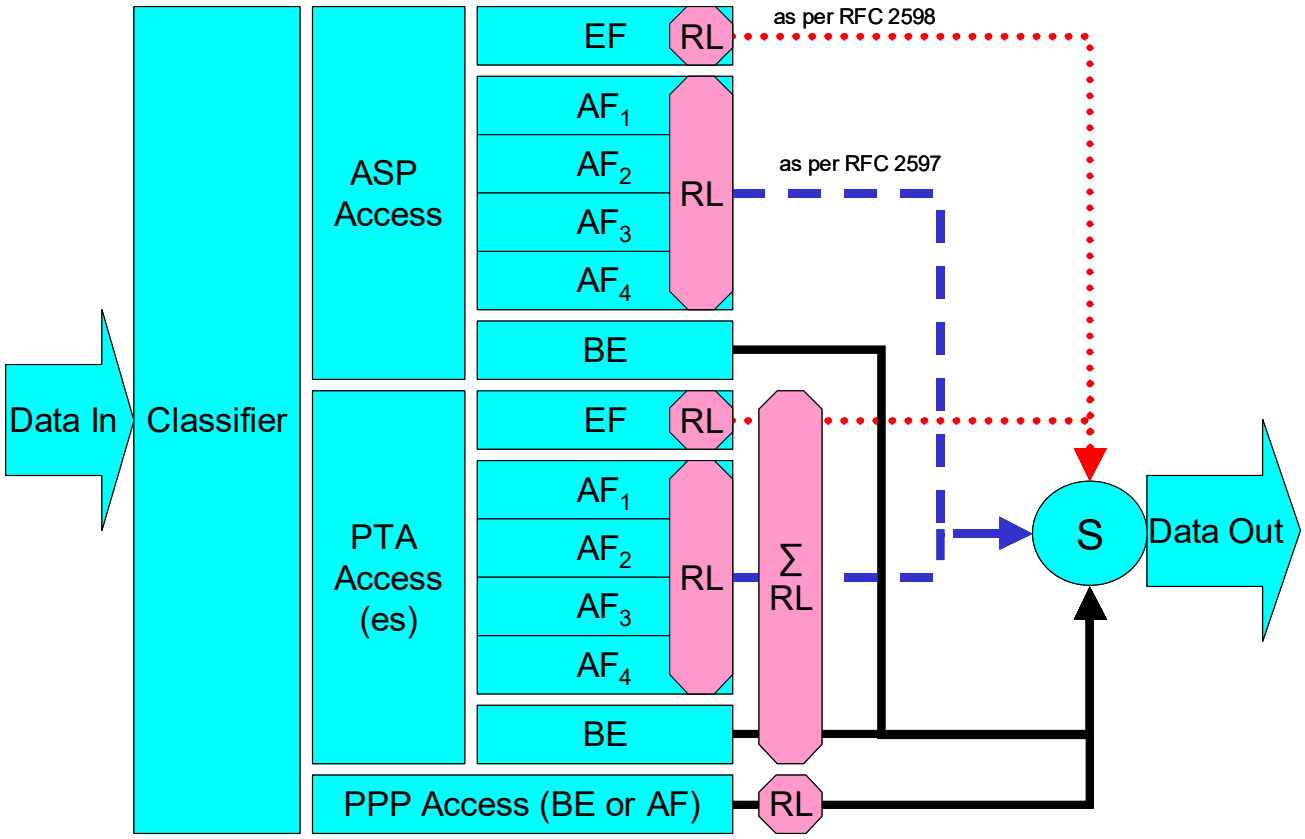



**Figure 24 – Queuing and Scheduling Example for RG**

In Figure 24 the following abbreviations apply:

ASP – Application Service Provider
PTA – PPP Terminated Aggregation
PPP – Point-to-Point Protocol
EF – Expedited Forwarding – as defined in RFC 3246 [11]
AF – Assured Forwarding – as defined in RFC 2597 [10]
BE – Best Effort forwarding
RL – Rate Limiter
∑RL – Summing Rate Limiter (limits multiple flows)
S – Scheduler

# Appendix II Use of Bridging Objects for VLAN Tagging

In the case of an Ethernet upstream Interface or a VDSL2 upstream Interface based on PTM-EFM, 802.1Q Tagging can be used to tag egress traffic. This choice enables a multi-VLAN architecture in order to deploy a multi-service configuration (high speed Internet, VoIP, Video Phone, IPTV, etc.), where one VLAN or a group of VLANs are associated with each service. If 802.1Q tagging on the upstream interface is used, it is necessary to have a way to associate incoming upstream 802.1Q tagged or untagged traffic or internally generated traffic (PPPoE, IPoE connections) to the egress (and vice-versa). The solution is to apply coherent bridging rules.

Regarding different traffic bridging rules, the possible cases are characterized as follows:

- Tagged LAN to tagged WAN traffic (pure VLAN bridging), with VLAN ID translation as a special case
- Untagged LAN to tagged WAN traffic
- Internally generated to tagged WAN traffic

To better understand the different cases, refer to Figure 25 and to the following examples.

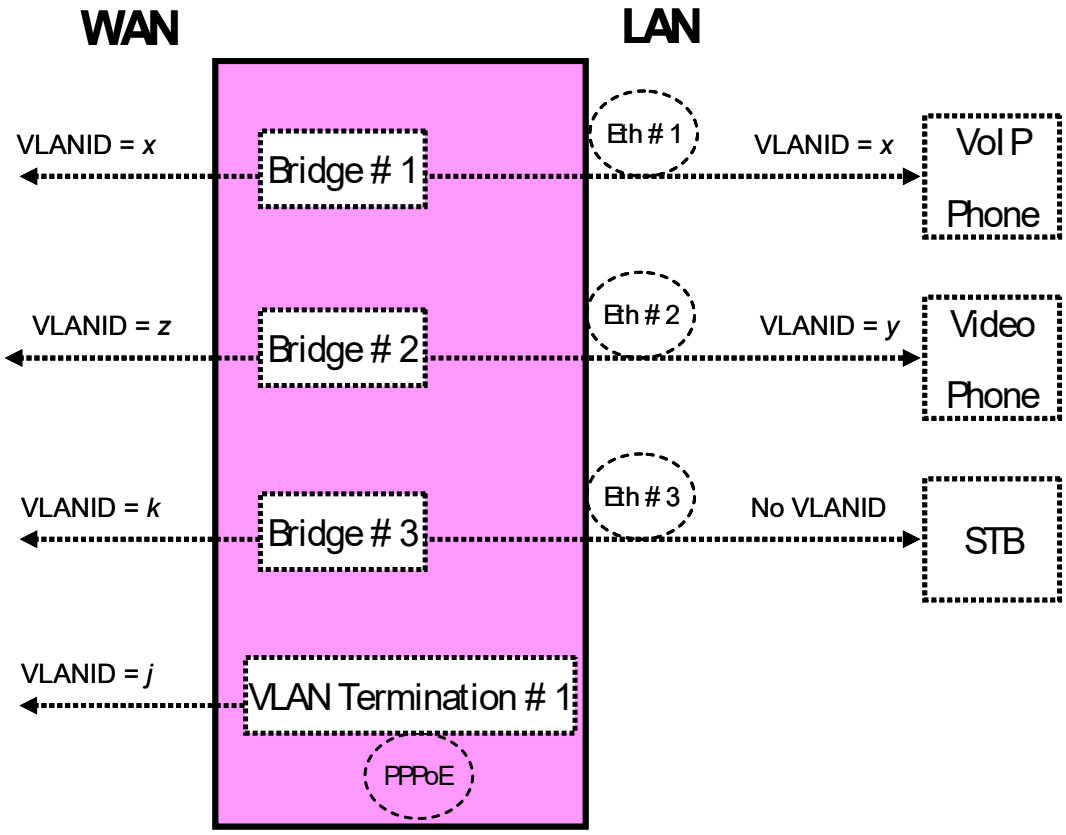



**Figure 25 – Examples of VLAN configuration based on Bridging and VLAN Termination objects**

## II.1 Tagged LAN to Tagged WAN Traffic (VLAN Bridging)

Ethernet port 1 (instance Device.Ethernet.Interface.2) might be dedicated to VoIP service, receiving VLAN ID x tagged traffic from a VoIP phone, and this port would be included in the same bridge dedicated to VoIP service on the upstream interface (instance Device.Ethernet.Interface.1), identified with the same VLAN ID x.

To achieve this, an interface-based bridge would be created using the Bridging object. A Bridge table entry would be created with entries for Ethernet port 1 and the upstream interface and for the VLAN ID x associated with VoIP.

The Bridging model is depicted in Figure 26, while the configuration rules for this situation are summarized in Table 6.

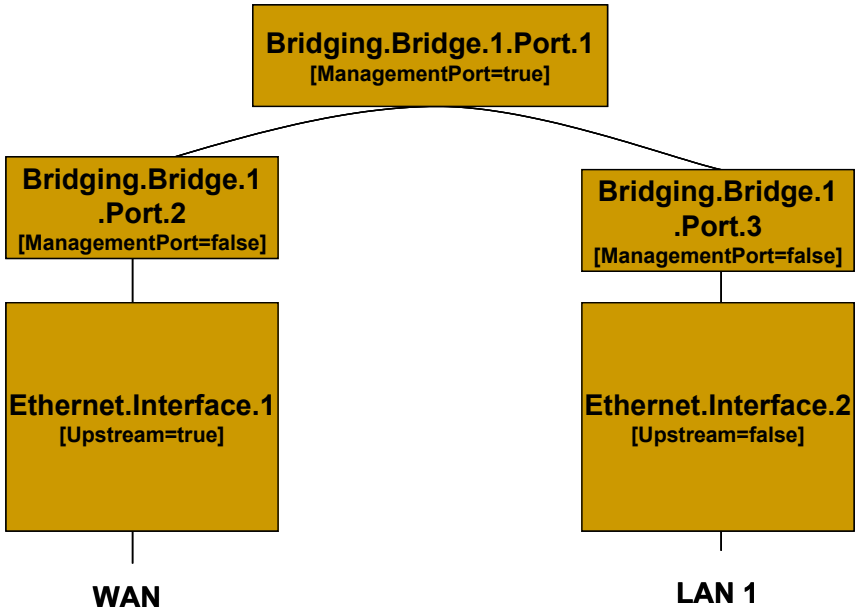



**Figure 26 – Bridge 1 model**

## Table 6 – Tagged LAN to tagged WAN configuration

<table>
<tr><th>Description</th><th>Bridging Configuration</th></tr>
<tr><td>Bridge between WAN and LAN 1 interfaces with VLANID=x</td><td>
[Define VLANx]
<table>
<tr><td>Device.Bridging.Bridge.1.VLAN.1</td><td>-</td></tr>
<tr><td>Name</td><td>VLANx</td></tr>
<tr><td>VLANID</td><td>X</td></tr>
</table>
[Define Ingress Port2-3 – Create an entry for the upstream and downstream port]:
<table>
<tr><td>Device.Bridging. Bridge.1.Port.2</td><td>-</td></tr>
<tr><td>PVID</td><td>x</td></tr>
<tr><td>Name</td><td>Port2</td></tr>
<tr><td>AcceptableFrameTypes</td><td>AdmitOnlyVLANTagged</td></tr>
</table>
<table>
<tr><td>Device.Bridging. Bridge.1.Port.3</td><td>-</td></tr>
<tr><td>PVID</td><td>x</td></tr>
<tr><td>Name</td><td>Port3</td></tr>
<tr><td>AcceptableFrameTypes</td><td>AdmitOnlyVLANTagged</td></tr>
</table>
[Associate Egress Port2-3 to VLANx - Create an entry for the upstream and downstream port]
<table>
<tr><td>Device.Bridging.Bridge.1.VLANPort.1</td><td>-</td></tr>
<tr><td>VLAN</td><td>VLANx</td></tr>
<tr><td>Port</td><td>Port2</td></tr>
<tr><td>Untagged</td><td>false</td></tr>
</table>
<table>
<tr><td>Device.Bridging.Bridge.1.VLANPort.2</td><td>-</td></tr>
<tr><td>VLAN</td><td>VLANx</td></tr>
<tr><td>Port</td><td>Port3</td></tr>
<tr><td>Untagged</td><td>false</td></tr>
</table>
</td></tr>
</table>

## II.2 Tagged LAN to Tagged WAN Traffic (Special Case with VLAN ID Translation)

Ethernet port 2 (instance Device.Ethernet.Interface.3) might be dedicated to Video Phone service, receiving VLAN ID y tagged traffic from a Video phone, and this port would be included in the same bridge dedicated to Video Phone service on the upstream interface (instance Device.Ethernet.Interface.1), identified by a different VLAN ID (VLAN ID z). In this case a VLAN translation needs to be performed.

To achieve this, a bridge would be created using the Bridging object. A Bridge table entry would be created along with two associated Filter object entries for {Ethernet port 2/VLAN ID z} and {upstream interface/VLAN ID y}. The Filter identifies the ingress interface and causes the ingress frames to be bridged to the egress VLAN, permitting VLAN-ID translation.

The Bridging model is depicted in Figure 27, while the configuration rules for this situation are summarized in Table 7.

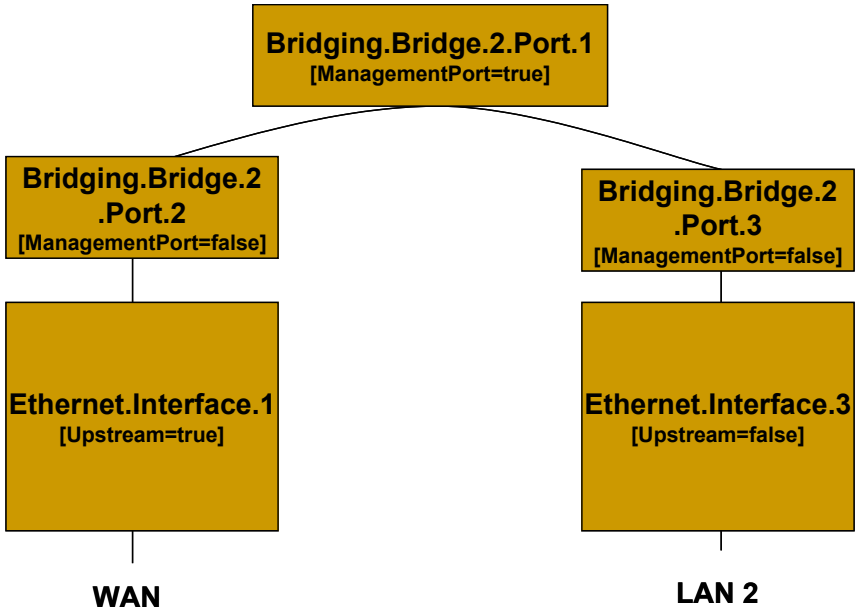



**Figure 27 – Bridge 2 model**

## Table 7 – Tagged LAN to tagged WAN configuration (VLAN ID translation)

<table>
<tr><th>Description</th><th>Bridging Configuration</th></tr>
<tr><td rowspan="4">Tagged LAN 2 to tagged WAN traffic (and vice versa) (special case with VLAN ID translation)<br>upstream VLAN-ID=z<br>downstream VLAN-ID=y</td><td>[Define VLANy and VLANz]<br><table><tr><td>Device.Bridging.Bridge.2.VLAN.1</td><td></td></tr><tr><td>Name</td><td>VLANy</td></tr><tr><td>VLANID</td><td>y</td></tr></table><br><table><tr><td>Device.Bridging.Bridge.2.VLAN.2</td><td></td></tr><tr><td>Name</td><td>VLANz</td></tr><tr><td>VLANID</td><td>z</td></tr></table></td></tr>
<tr><td>[Define Ingress Port2 – Create an entry for upstream port]:<br><table><tr><td>Device.Bridging.Bridge.2.Port.2</td><td></td></tr><tr><td>PVID</td><td>Z</td></tr><tr><td>Name</td><td>Port2</td></tr><tr><td>AcceptableFrameTypes</td><td>AdmitOnlyVLANTagged</td></tr></table><br>[Define Ingress Port3 – Create an entry for the downstream port]:<br><table><tr><td>Device.Bridging.Bridge.2.Port.3</td><td></td></tr><tr><td>PVID</td><td>y</td></tr><tr><td>Name</td><td>Port3</td></tr><tr><td>AcceptableFrameTypes</td><td>AdmitOnlyVLANTagged</td></tr></table></td></tr>
<tr><td>[Associate Egress Port2 to VLANz - Create an entry for upstream port]<br><table><tr><td>Device.Bridging.Bridge.2.VLANPort.1</td><td>-</td></tr><tr><td>VLAN</td><td>VLANz</td></tr><tr><td>Port</td><td>Port2</td></tr><tr><td>Untagged</td><td>false</td></tr></table><br>[Associate Egress Port3 to VLANy - Create an entry for each downstream port]<br><table><tr><td>Device.Bridging.Bridge.2.VLANPort.2</td><td>-</td></tr><tr><td>VLAN</td><td>VLANy</td></tr><tr><td>Port</td><td>Port3</td></tr><tr><td>Untagged</td><td>false</td></tr></table></td></tr>
</table>

<table>
<tr><th>Description</th><th>Bridging Configuration</th></tr>
<tr><td></td><td>[Define filter on upstream: ingress from Port 2 is associated with VLANy]
<table>
<tr><td>Device.Bridging.Filter.1.</td><td>-</td></tr>
<tr><td>Bridge</td><td>VLANy</td></tr>
<tr><td>Interface</td><td>Port2</td></tr>
</table>
[Define filter on downstream: ingress from Port 3 is associated with VLANz]
<table>
<tr><td>Device.Bridging.Filter.2.</td><td>-</td></tr>
<tr><td>Bridge</td><td>VLANz</td></tr>
<tr><td>Interface</td><td>Port3</td></tr>
</table>
</td></tr>
</table>

## II.3 Untagged LAN to Tagged WAN Traffic

Ethernet port 3 (instance Device.Ethernet.Interface.4) might be dedicated to IPTV service, receiving untagged traffic from a STB, and this port would be included in the same bridge dedicated to IPTV service on the upstream interface (instance Device.Ethernet.Interface.1), identified with the VLAN ID k.

To achieve this, an interface-based bridge would be created using the Bridging object. A Bridge table entry would be created, associating in the same bridge untagged frames on Ethernet port 3 with tagged frames on the upstream interface.

The Bridging model is depicted in Figure 28, while the configuration rules for this situation are summarized in Table 8.

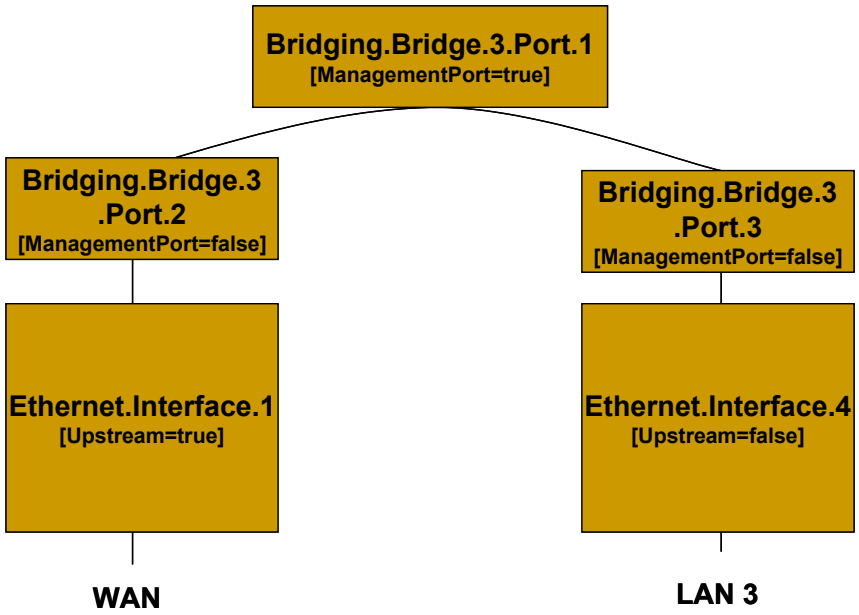



**Figure 28 – Bridge 3 model**

**Table 8 – Untagged LAN to tagged WAN configuration**

<table>
<tr><th>Description</th><th>Bridging Configuration</th></tr>
<tr><td rowspan="3">Untagged LAN 3 to tagged WAN (VLAN-ID=k) traffic</td><td>[Define VLANk]<br><table><tr><td>Device.Bridging.Bridge.3.VLAN.1</td><td></td></tr><tr><td>Name</td><td>VLANk</td></tr><tr><td>VLANID</td><td>k</td></tr></table></td></tr>
<tr><td>[Define Ingress Port2 – Create an entry for upstream port]:<br><table><tr><td>Device.Bridging.Bridge.3.Port.2</td><td></td></tr><tr><td>PVID</td><td>k</td></tr><tr><td>Name</td><td>Port2</td></tr><tr><td>AcceptableFrameTypes</td><td>AdmitOnlyVLANTagged</td></tr></table>[Define Ingress Port3 – Create an entry for the downstream port]:<br><table><tr><td>Device.Bridging.Bridge.3.Port.3</td><td></td></tr><tr><td>Name</td><td>Port3</td></tr><tr><td>AcceptableFrameTypes</td><td>AdmitAll</td></tr></table></td></tr>
<tr><td>[Associate Egress Port2 to VLANk - Create an entry for upstream port]<br><table><tr><td>Device.Bridging.Bridge.3.VLANPort.1</td><td>-</td></tr><tr><td>VLAN</td><td>VLANk</td></tr><tr><td>Port</td><td>Port2</td></tr><tr><td>Untagged</td><td>false</td></tr></table>[Associate Egress Port3 to VLANk - Create an entry for each downstream port]<br><table><tr><td>Device.Bridging.Bridge.3.VLANPort.2</td><td>-</td></tr><tr><td>VLAN</td><td>VLANk</td></tr><tr><td>Port</td><td>Port3</td></tr><tr><td>Untagged</td><td>true</td></tr></table></td></tr>
</table>

## II.4 Internally Generated to Tagged WAN Traffic

A CPE PPPoE internal session (instance Device.PPP.Interface.1) might be dedicated to Management service and this logical interface would encapsulate/de-encapsulate its outgoing or incoming traffic in the VLAN ID j, dedicated to Management service.

To achieve this, instead of using a bridging object, a VLAN Termination interface would be created (Device.Ethernet.VLANTermination.1). The Bridging model is depicted in Figure 29, while the configuration rules for this situation are summarized in Table 9.

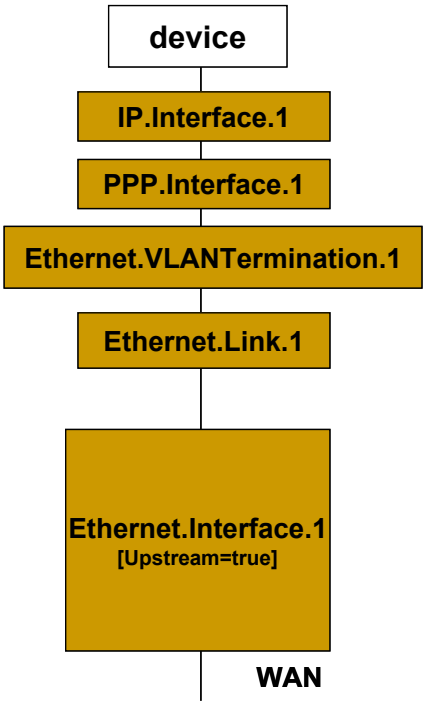



**Figure 29 – VLAN Termination model**

**Table 9 – Internally generated to tagged WAN configuration**

| Description | VLAN Termination Configuration |
|---|---|
| Internal to tagged WAN (VLAN-ID=j) traffic | [DefineVLAN Termination on top of Ethernet Link]<br><br>Device.Ethernet.VLANTermination.1 \| <br>VLANID \| *j*<br>LowerLayers \| Ethernet.Link.1 |

## II.5 Other Issues

The previous rules can be applied to allow all combinations of traffic. If the subscriber's services are modified, the Bridging configuration might need to be modified accordingly.

It can be interesting to detail the configuration of three special cases:

- More than one downstream interface in a bridge
- 802.1D (re-)marking
- More than one VLAN ID tag for the same downstream interface

### II.5.1 More than one Downstream Interface in a Bridge

Referring to the example in Section II.1, Tagged LAN to tagged WAN traffic (VLAN bridging), consider adding other Ethernet interfaces (e.g. Ethernet ports 3 and 4 = instance Device. Ethernet.Interface.3/4) to the Video Phone service. The behavior is the same as for the existing Ethernet port 2 (instance Device.Ethernet.Interface.2).

To achieve this, new entries need to be added for interface Eth-3 and Eth-4. The Bridging model is depicted in Figure 30, while the configuration rules for this situation are summarized in Table 6 and Table 10.

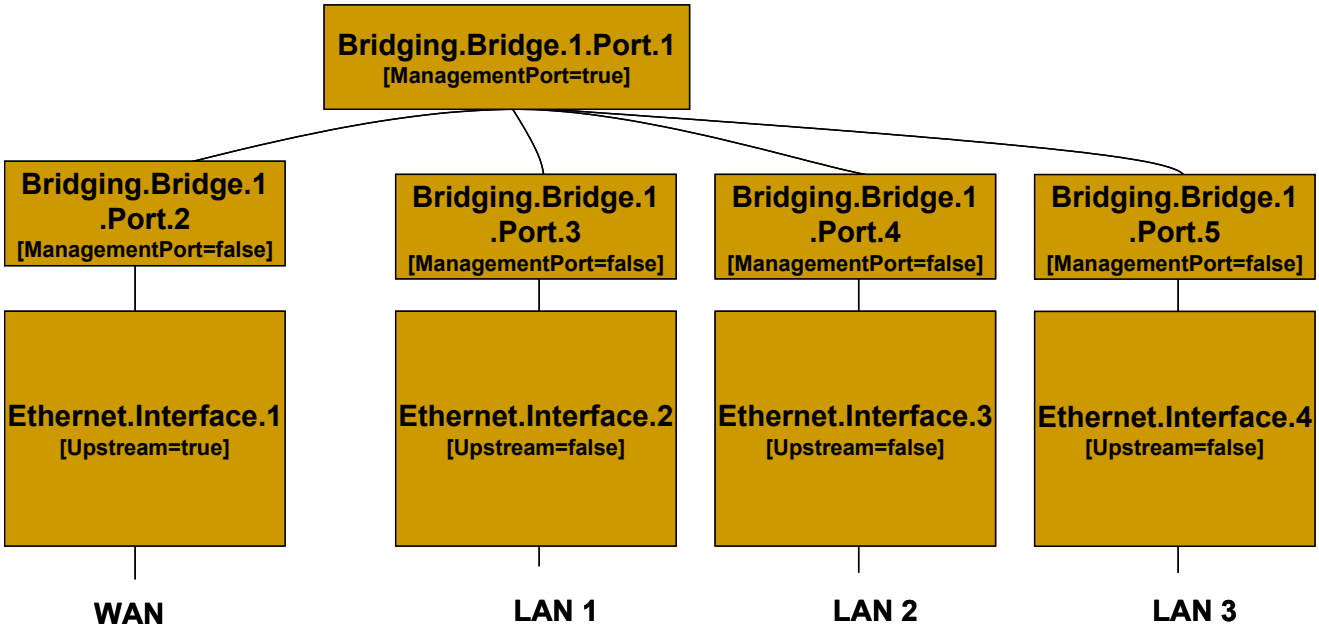



**Figure 30 – Bridge 1 model**

**Table 10 – Configuration to be added to Table 6**

<table>
<tr><th>Description</th><th>Bridging Configuration</th></tr>
<tr><td>Bridge between WAN and LAN 2/LAN 3 interfaces with VLANID=x<br><br>(Configuration to be added to Table 6)</td><td>[Define Ingress Port4-5 – Create an entry for the other downstream ports]:<br><br>
<table>
<tr><td>Device.Bridging. Bridge.1.Port.4</td><td>-</td></tr>
<tr><td>PVID</td><td>x</td></tr>
<tr><td>Name</td><td>Port4</td></tr>
<tr><td>AcceptableFrameTypes</td><td>AdmitOnlyVLANTagged</td></tr>
</table>
<br>
<table>
<tr><td>Device.Bridging. Bridge.1.Port.5</td><td>-</td></tr>
<tr><td>PVID</td><td>x</td></tr>
<tr><td>Name</td><td>Port5</td></tr>
<tr><td>AcceptableFrameTypes</td><td>AdmitOnlyVLANTagged</td></tr>
</table>
<br>[Associate Egress Port4-5 to VLANx - Create an entry for the downstream ports]<br><br>
<table>
<tr><td>Device.Bridging.Bridge.1.VLANPort.3</td><td>-</td></tr>
<tr><td>VLAN</td><td>VLANx</td></tr>
<tr><td>Port</td><td>Port4</td></tr>
<tr><td>Untagged</td><td>false</td></tr>
</table>
<br>
<table>
<tr><td>Device.Bridging.Bridge.1.VLANPort.4</td><td>-</td></tr>
<tr><td>VLAN</td><td>VLANx</td></tr>
<tr><td>Port</td><td>Port5</td></tr>
<tr><td>Untagged</td><td>false</td></tr>
</table>
</td></tr>
</table>

## II.5.2 802.1D (Re)-marking

The 802.1Q Tag includes the 802.1D user priority bits field. All the previous cases can also be extended to mark (or re-mark) this 802.1D field. To achieve this, there are different configuration options; one of them is to use the DefaultUserPriority or PriorityRegeneration fields in the Bridge Port object. For untagged frames, more complex rules can be defined referring to the QoS Classification, using the PriorityTagging value. The Bridging configuration

rules for marking egress traffic on the upstream interface are summarized in Table 11. Compare it with Table 6.

**Table 11 – 802.1D (re-)marking**

<table>
<tr><th>Description</th><th>Bridging Configuration</th></tr>
<tr><td rowspan="3">802.1D (re-)marking<br>Remark all WAN egress traffic</td><td>[Mark the ingress frames with Default user Priority, in this case 0]<br><table><tr><td>Device.Bridging. Bridge.1. Port.2.</td><td></td></tr><tr><td>DefaultUserPriority</td><td>0</td></tr></table><br>[Remark each ingress priority value (0,1,2,3,4,5,6,7) with the priority regeneration string, in this case (0,0,0,0,4,4,4,4)]<br><table><tr><td>Device.Bridging. Bridge.1. Port.2.</td><td></td></tr><tr><td>PriorityRegeneration</td><td>0,0,0,0,4,4,4,4</td></tr></table></td></tr>
<tr><td>[In case of ingress untagged frames, for more complex classification, QoS object are referred. In this case remark with 0]<br><table><tr><td>Device.Bridging. Bridge.1. Port.2.</td><td></td></tr><tr><td>PriorityTagging</td><td>true</td></tr></table></td></tr>
<tr><td><table><tr><td>Device.QoS. Classification. {i}.</td><td></td></tr><tr><td>EthernetPriorityMark</td><td>0</td></tr></table></td></tr>
</table>

### II.5.3 More than one VLAN ID Tag Admitted on the Same Downstream Interface

Another scenario that can be further detailed is the case of more than one VLAN ID tag admitted on the same downstream interface. A practical example would be a 2 box scenario, with a User Device generating traffic segregated in multiple VLANs (e.g. a router offering services to the customer), and a Residential Gateway, providing upstream connectivity to the Access Network, with the connection between the two pieces of equipment using an Ethernet interface.

In this case, we assume the User Device is able to tag the different traffic flows, segregating the different services (Voice, Video, …) into different VLANs. The Residential Gateway needs, on the same downstream interface, to be able to receive different VLAN ID and correctly forward or translate to the upstream interface (and vice versa). To achieve this, appropriate Bridging objects need to be configured.

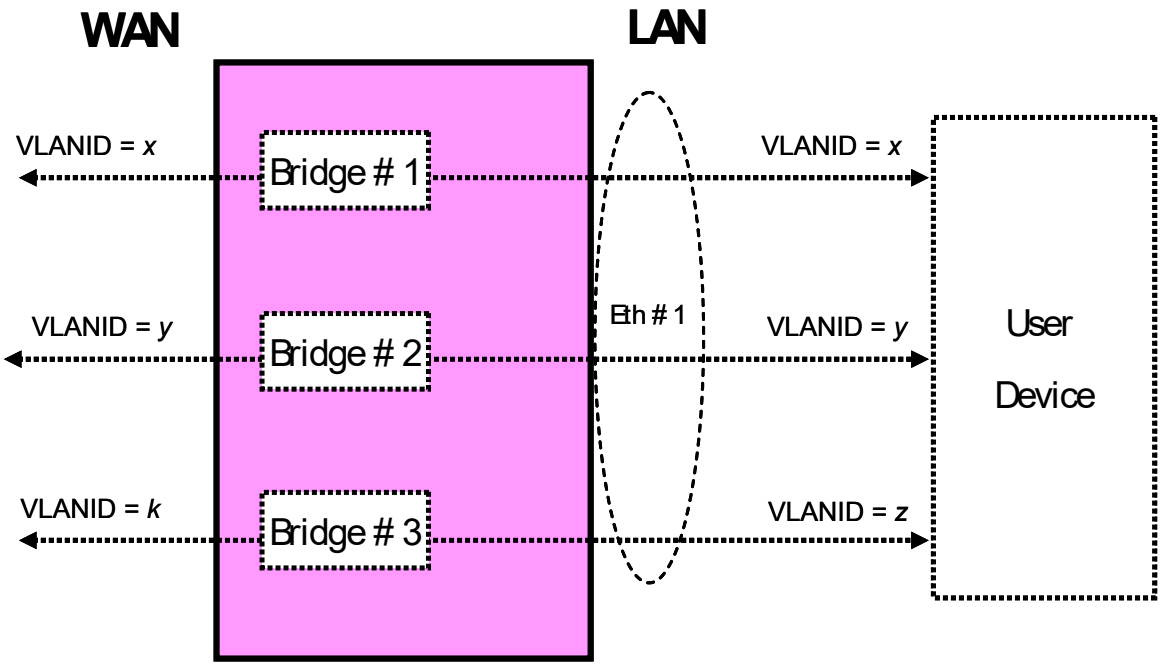



**Figure 31 – Example of VLAN configuration in a 2 box scenario**

**Referring to**

Figure 31 as an example, assume the case of three VLANs (VLAN ID=x,y,z) offered by a User Device to the Residential Gateway on the same downstream interface (Eth #1). The Residential Gateway bridges two of them (VLAN ID=x,y) and translates the other one (VLAN ID=z) to the upstream interface (VLAN ID=k).

On the Residential Gateway, this can be achieved using a combination of the Bridging objects detailed in the preceding sections, with 3 bridge entries and their related entries. Refer to Figure 32 for the Bridging model and Table 12 for the global configuration.

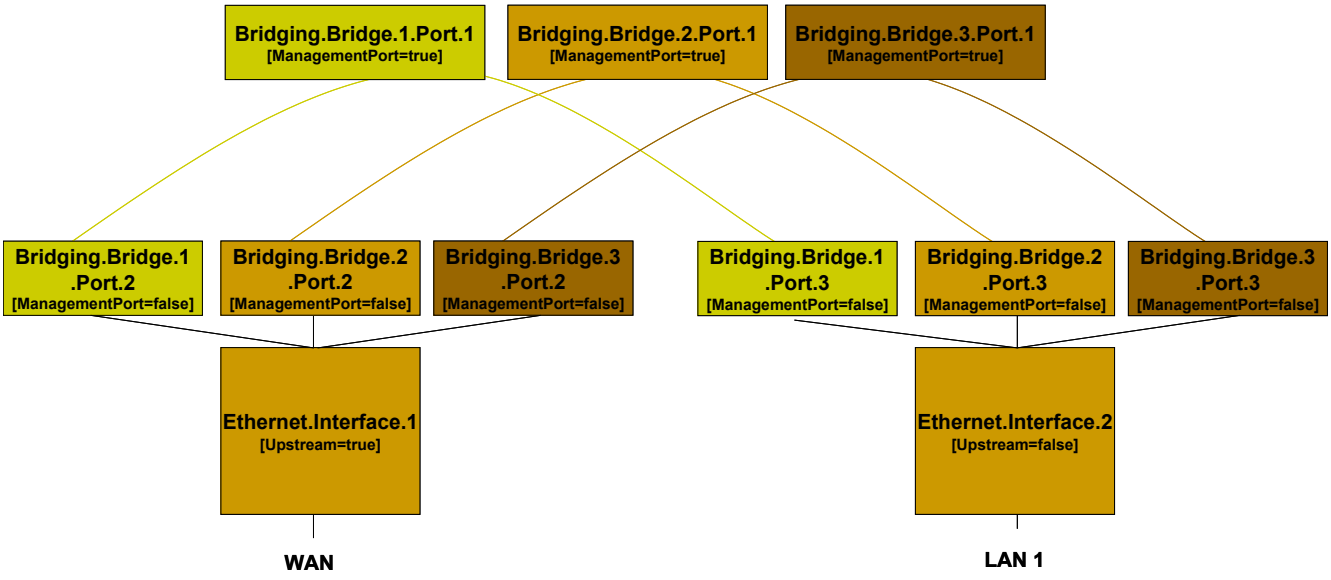


**Figure 32 – Bridge 1,2,3 model**

**Table 12 – More than one VLAN ID tag admitted on the same Downstream interface**

<table>
<tr><th>Description</th><th>Bridging Configuration</th></tr>
<tr><td>More than one VLAN ID tag admitted on the same downstream interface</td><td>The configuration is the sum of Sections II.1 and II.2, but on the downstream side the lower layer to be configured for each Bridge Port is always: Ethernet.Interface.2<br><br>
<table>
<tr><td>Device.Bridging. Bridge.1. Port.3.</td><td></td></tr>
<tr><td>LowerLayers</td><td>Ethernet.Interface.2</td></tr>
</table>
<table>
<tr><td>Device.Bridging. Bridge.2. Port.3.</td><td></td></tr>
<tr><td>LowerLayers</td><td>Ethernet.Interface.2</td></tr>
</table>
<table>
<tr><td>Device.Bridging. Bridge.3. Port.3.</td><td></td></tr>
<tr><td>LowerLayers</td><td>Ethernet.Interface.2</td></tr>
</table>
</td></tr>
</table>

# Appendix III Wi-Fi Theory of Operation

This section discusses the theory of operations for various technologies in the Wi-Fi domain found within the Device:2 data model.

## III.1 Multi-radio and Multi-band Wi-Fi Radio Devices

The WiFi.Radio object description says "This object models an 802.11 wireless radio on a device. If the device can establish more than one connection simultaneously (e.g. a dual radio device), a separate WiFi.Radio instance will be used for representing each physical radio of the device."

The following sections clarify when multiple WiFi.Radio instances are needed, and the impact on their underlying parameters in the case of multi-radio and/or multi-band devices.

## III.2 Definitions

Each physical radio allows the transmission and reception of data on a single Wi-Fi channel at a given time. A single-radio device is able to transmit/receive a packet at a given time only on one Wi-Fi channel. Similarly, a dual-radio device is able to simultaneously transmit/receive data on two Wi-Fi channels. In general, a device with N radios is able to simultaneously transmit/receive data on N Wi-Fi channels.

An important point is that Wi-Fi can operate at two different frequency bands, 2.4 GHz and 5 GHz, as follows:

- Wi-Fi technologies based on IEEE 802.11b/g standard operate on the 2.4 GHz frequency band.
- Wi-Fi technologies based on IEEE 802.11a/ac standard operate on the 5 GHz frequency band.
- Wi-Fi technologies based on IEEE 802.11n standard operate on both the 2.4 and 5 GHz frequency bands.

Radios that operate at a single frequency band (e.g. 2.4 GHz only 802.11b/g devices) are called single-band radios. Radios that can operate at both 2.4 and 5 GHz frequency bands (e.g. 802.11a/b/g/n/ac devices) are called dual-band radios.

A dual-band device can be a single-radio device if it can be configured to operate at 2.4 or 5 GHz frequency bands. However, only a single frequency band is used to transmit/receive at a given time. In such a case the device has a single physical radio that is dual-band.

Also, a dual-radio single-band device can exist (although uncommon) if both radios are single-band.

## III.3 Number of Instances of WiFi.Radio Object

Given the definitions above, a separate WiFi.Radio instance will be used for each physical radio of the device, i.e. one instance for a single-radio device, two instances for dual-radio devices, and so on. A single WiFi.Radio instance will therefore be used for a dual-band single-radio device.

Each WiFi.Radio instance is configured separately and is, in general, completely independent of other instances.

## III.4 SupportedFrequencyBands and OperatingFrequencyBand

The frequency band used by a WiFi device is an important parameter. With first generations of WiFi technologies, the specific frequency band was linked to the IEEE standard in use (i.e. 802.11b/g are 2.4 GHz standards, while 802.11a is a 5 GHz standard). With the introduction of the IEEE 802.11n standard, which can work both at 2.4 and 5 GHz, two specific parameters are used to indicate the supported frequency bands and the operating frequency band.

SupportedFrequencyBands is a list-valued parameter, containing one item for single-band radios (either 2.4GHz or 5GHz) and two items for dual-band radios (both 2.4GHz and 5GHz).

The OperatingFrequencyBand parameter specifies which frequency band is currently being used by a dual-band radio (i.e. set to one of the two items listed in the SupportedFrequencyBands parameter). For single-band radios, OperatingFrequencyBand always has the same value as SupportedFrequencyBands (since only one frequency band is supported).

## III.5 Behavior of Dual-band Radios when OperatingFrequencyBand Changed

When the configured operating frequency band of a dual-band radio is changed (i.e. the value of the OperatingFrequencyBand parameter is modified), this has an impact on other parameters within the WiFi.Radio object.

The Channel parameter has to be changed, since channels for the 2.4 GHz frequency band are in the range 1-14, while channels for the 5 GHz frequency band can be in the range of 36-165 (for example). The expected behavior is that, upon modifying the OperatingFrequencyBand parameter, the device automatically selects a new channel that is valid for the new frequency band (according to some vendor-specific selection procedure).

Other related parameters of significance for the Channel properties are AutoChannelEnable, OperatingChannelBandwidth and CurrentOperatingChannelBandwidth.

Persistence of the Channel parameter value for the previous frequency band is not required. For example, if OperatingFrequencyBand is later changed back to *5GHz*, a new valid value for the Channel parameter is automatically selected by the device, but this value need not be the same as was selected the last time OperatingFrequencyBand was set to *5GHz*.

Other parameters whose values can be impacted when the OperatingFrequencyBand changes, include: ExtensionChannel, PossibleChannels, SupportedStandards, OperatingStandards, IEEE80211hSupported, and IEEE80211hEnabled. If the current value is no longer valid, the device will automatically select a valid new value according to some vendor-specific procedure, and the old value need not persist.

## III.6 SupportedStandards and OperatingStandards

The SupportedStandards parameter is a list of all IEEE 802.11 physical layer modes supported by the devices. Wi-Fi is in general backward compatible, so 802.11g devices are also 802.11b devices, 802.11n devices are also 802.11b/g devices (if operating at 2.4 GHz), and 802.11n devices are also 802.11a devices (if operating at 5 GHz).

For dual-band radios, the OperatingFrequencyBand parameter is used for switching the operating frequency band. For this reason SupportedStandards only includes those values corresponding to operation in the frequency band indicated by the OperatingFrequencyBand parameter. For example, for dual-band 802.11a/b/g/n devices, SupportedStandards can be *b, g, n* when OperatingFrequencyBand is *2.4GHz* and *a, n, ac* when OperatingFrequencyBand is *5GHz*.

The OperatingStandards parameter is used to limit operation to a subset of physical modes supported. For example, an 802.11b/g/n radio will have *b, g, n* value for the SupportedStandards parameter, but can be configured to operate only with 802.11n by setting the OperatingStandards parameter to *n*.

## III.7 Different Types of WiFi Errors

This section first describes the different WiFi data units and the layers where they apply.

The MAC Service Data Unit (MSDU) is the service data unit that is received from the logical link control (LLC) sub-layer which lies above the medium access control (MAC) sub-layer in the protocol stack.

The MAC protocol data unit (MPDU) is a message exchanged between MAC entities in a communication system. “WiFi frames” refer to MPDUs and WiFi counters are counts of MPDUs.

The Physical Layer Convergence Procedure (PLCP) protocol data unit (PPDU) corresponds with the bits that are actually transmitted across the physical layer.

The MSDU is the frame that interfaces to higher layers, while the MPDU is the frame that is actually transmitted through the wireless medium, excluding the physical layer overhead. The MPDU is the MSDU plus MAC layer overhead (header, FCS, etc.). The PPDU is the MPDU plus physical layer overhead (preamble, PHY header, etc.).

The number of errored MPDUs is the number of MPDUs without corresponding ACKs. In most cases, the number of MSDUs is the same as the number of MPDUs. However, if fragmentation is

enabled, then one MSDU can become multiple MPDUs, and there is one ACK per MPDU, hence multiple ACKs for one MSDU.

With frame aggregation in 802.11n, multiple MPDUs become one aggregated MPDU (A-MPDU). There is usually one block ACK for each A-MPDU, and only the errored MPDU(s) can be retransmitted selectively. In this case the WiFi counters will count the original MPDUs and not the A-MPDUs.

To avoid confusion that may be caused by fragmentation or frame aggregation, "WiFi frames" or packets are all considered here to be MPDUs and WiFi counters refer to MPDUs.

Figure 33 explains the process of the MSDU/MPDU flow structure through the MAC layer of the WiFi receiver.

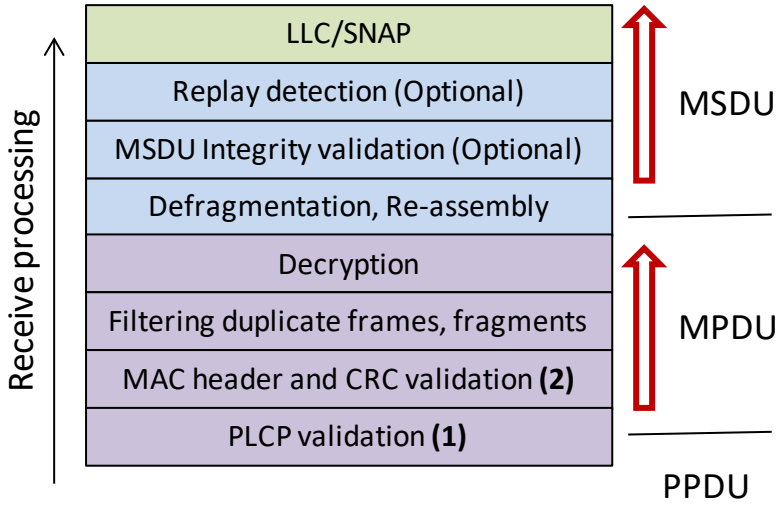



**Figure 33 – WiFi functions within layers**

PLCPErrorCount: This error occurs at point (1) in Figure 33, and is the first error type that can be counted. The PLCPErrorCount is the number of errors in the PLCP headers of the received MPDUs, which is the number of frames for which the parity check of the PLCP header failed.

There are two errors that happen at point (2) of the wireless reception:

FCSErrorCount: This error occurs at point (2) in Figure 33. After the MPDU passes the PLCP header check, it is passed onto MAC layer validation. The FCSErrorCount is the number of frames for which the Frame Check Sequence (FCS) at the end of the MAC frame was in error.

InvalidMACCount: This error also occurs at point (2) in Figure 33. The MAC header of the MPDU has a field called 'Protocol Version'. Currently, it is set to '0'. If this number is anything but 0, or the frame type is not data/control/management,' the InvalidMACCount is incremented.

After verifying that the frame was received without errors, the WiFi receiver will then check if the frame was designated for its own use or not (still MAC layer).

PacketsOtherReceived: This counter is used to catch those MPDUs that are not addressed to this radio. This can be assessed by checking if the 'Address 1' field of the 802.11 MAC header

contains a MAC address that is associated with this radio, if not then ‘PacketsOtherReceived’ is incremented.

After this step, the AP can also discard duplicate frames or fragments among the frames addressed to it, to simplify higher-level processing.

The ErrorsReceived count is the sum of the PLCPErrorCount plus the FCSErrorCount plus the InvalidMACCount.

# Appendix IV Use Cases

This section presents a number of management-related use cases that correspond to typical Controller activities.

## IV.1 Create a WAN Connection

The Controller can create the objects in the interface stack bottom-up. Each time a new higher-layer object is created, the link with the underlying interface object needs to be set. The layer 1 interface, in this case a DSL.Channel and DSL.Line object, will already exist (A Controller cannot create physical interfaces).

1. The Controller creates a new ATM.Link object, a new Ethernet.Link object, a new PPP.Interface object, and a new IP.Interface object.
2. The LowerLayers parameter in an existing DSL.Channel object is already linked to an existing DSL.Line object (A Controller cannot configure this linkage).
3. The Controller configures the new objects including enabling the objects and using the LowerLayers parameters as follows:
    a. Setting the LowerLayers parameter in the ATM.Link object to link it to an existing DSL.Channel object that is configured with ATM encapsulation (i.e. the read-only LinkEncapsulationUsed parameter in the DSL.Channel object is set to one of the ATM-related enumeration values).
    b. Setting the LowerLayers parameter in the Ethernet.Link object to link it to the ATM.Link object.
    c. Setting the LowerLayers parameter in PPP.Interface object to link it to the Ethernet.Link object.
    d. Setting the LowerLayers parameter in IP.Interface object to link it to the PPP.Interface object.
4. The CPE updates the InterfaceStack table automatically. The stack looks like this: IP.Interface → PPP.Interface → Ethernet.Link → ATM.Link → DSL.Channel → DSL.Line.
5. Note that the Controller might also want to update other related objects, including the NAT object, the Routing.Router object, or various QoS and Bridging tables. VLANs might also need to be created.

## IV.2 Modify a WAN Connection

In this use case, the Controller needs to modify an existing WAN connection, in order to insert a new layer in the stack or to change some portion of the interface stack. This is not the management WAN connection. For the purposes of this example, the Controller is changing the WAN connection in use case IV.1 to make use of PTM rather than ATM-based aggregation.

1. The Controller creates a new PTM.Link object.
2. The Controller configures the objects, including enabling the new PTM.Link object and using the LowerLayers parameter as follows:
    a. Setting the LowerLayers parameter in the PTM.Link object to link it to an existing DSL.Channel object that is configured with PTM encapsulation (i.e. the

read-only LinkEncapsulationUsed parameter in the DSL.Channel object is set to one of the PTM-related enumeration values).
   b. Setting the LowerLayers parameter in the Ethernet.Link object to refer to the PTM.Link object rather than the ATM.Link object.
   c. Setting the LowerLayers parameter in the IP.Interface object to refer to the Ethernet.Link object rather than the PPP.Interface object.
3. The CPE updates the InterfaceStack table automatically. The stack looks like this: IP.Interface → Ethernet.Link → PTM.Link → DSL.Channel → DSL.Line.
4. Note that the Controller might also want to update other related objects, including the Bridging table. The Controller might also want to delete the existing PPP.Interface and ATM.Link objects.

## IV.3 Delete a WAN Connection

Assume that we want to delete the WAN connection as it is configured in use case IV.1.

1. The Controller deletes the IP.Interface object.
2. The Controller deletes the PPP.Interface object.
3. The Controller deletes the Ethernet.Link object.
4. As each of these objects is deleted, the InterfaceStack is adjusted automatically by the CPE.
5. Any strong references to the deleted objects, e.g. in Device.QoS classification rules, will automatically be set to empty strings.

## IV.4 Discover whether the Device is a Gateway

Many operators want to determine if a particular device is a "gateway" or not. The term "gateway", however, is rather vague; usually the operator wants to know one (or more) of the following things:

1. If the device terminates the WAN connection(s).
2. If the device is responsible for providing DHCP addresses to the other devices in the home.
3. If the device provides functionality such as NAT or routing capabilities.

In order to determine if the device terminates a WAN connection, the Controller might look for an interface object with a technology that is by definition WAN (such as DSL) or for a technology that could be a WAN termination technology (such as Ethernet or MoCA).

In order to determine if the device is responsible for providing addresses to other devices in the home, the Controller could check for the existence of the DHCP Server object. The existence of the Host table also indicates that the device is aware of Hosts, by whatever means they're addressed.

For CWMP managed CPEs, the existence of the ManageableDevice table within the ManagementServer object also indicates that the device serves as the DHCP server for the TR-

069 managed device exchange defined in TR-069 [2] Annex F, which is also often an indication of "gateway" functionality.

In order to determine if the device provides functionality such as NAT or a router, the Controller would check for the existence of an enabled NAT or Routing.Router object.

## IV.5 Provide Extended Home Networking Topology View

Another use case is to determine the topology of the home network behind the gateway. For a generic understanding of the network, the Host table provides information such as the layer 2 and layer 3 interfaces via which the Host is connected as well as DHCP lease information for each connected Host.

If the operator is interested in UPnP devices in the home network, the UPnP.Discovery tables (RootDevice, Device, and Service) provide that information in addition to the Host table entries that correspond to a particular UPnP Root Device, Device, or Service.

Finally for CWMP enabled CPEs, the ManageableDevice table within the ManagementServer object provides information about the CWMP managed devices that the CPE has learned about through the DHCP message exchange defined in TR-069 [2] Annex F.

## IV.6 Determine Current Interfaces Configuration

One of the most fundamental Controller tasks is to determine the general picture of the interfaces for a device so that it can understand which WAN and LAN side connections exist.

In the Device:2 data model managed with CWMP, it would work this way:

1. The ACS would issue a GetParameterValues for the InterfaceStack table. This table would provide a list of all the Interface connections. The ACS could use this table to build up the general picture of the Interfaces that are part of the current configuration.
2. If the ACS is interested in the specifics of an individual interface, it can then go and issue GetParameterNames or GetParameterValues for the interfaces of interest.

If the CPE is managed by USP:

1. The USP Controller would issue a Get request for the InterfaceStack table. This table would provide a list of all the Interface connections. The USP Controller could use this table to build up the general picture of the Interfaces that are part of the current configuration.
2. If the USP Controller is interested in the specifics of an individual interface, it can then go and issue a filtered Get request message for the interfaces of interest.

## IV.7 Create a WLAN Connection

In this use case the Controller creates a new WLAN connection. For the purposes of illustration, in this example the Controller will create a new SSID object to link to an existing radio (a new SSID object implies a different SSID value than those used by existing WiFi connections). The layer 1 interface, in this case a WiFi.Radio object, will already exist (Controller can not create physical interfaces).

1. The Controller creates a new WiFi.SSID object and WiFi.AccessPoint object.
2. The Controller configures the new WiFi.SSID object, including enabling it and setting the value of the LowerLayers parameter to reference the device's WiFi.Radio object.
3. The Controller adds the new WiFi.SSID object to the LowerLayers parameter of an existing non-management Bridging.Bridge.{i}.Port object, as appropriate.

   *NOTE - A non-management bridge port is indicated when its ManagementPort parameter is set to false.*

4. The Controller configures the new WiFi.AccessPoint object, including enabling it and sets the value of its SSIDReference parameter to reference the WiFi.SSID object.
5. The CPE updates the InterfaceStack table automatically.
6. Note that the Controller might also want to update other related objects; also, if there were no appropriate existing bridge port to which to connect the SSID, the Controller might need to create that object as well.

## IV.8 Delete a WLAN Connection

In this use case the Controller deletes the SSID created in use case IV.7.

1. The Controller deletes the WiFi.SSID object and the WiFi.AccessPoint object.
2. The CPE automatically updates the InterfaceStack table.
3. Note that if the radio has no other SSIDs configured, this would operationally disable the wireless interface.

## IV.9 Configure a DHCP Client and Server

In this use case, the Controller wants to configure a DHCP server to provide private 192.168.1.x IP addresses to most home network (HN) devices, but to obtain IP addresses from the network for HN devices that present an option 60 (vendor class ID) value that begins with "ACME".

The ACME devices are remotely managed, so the Controller will also configure the DHCP clients on those devices and the DHCP server on the gateway.

### IV.9.1 DHCP Client Configuration (ACME devices)

The ACME devices are quite simple. Each has a single wired Ethernet port and a single IP interface.

A DHCP Client object is created and configured as follows:

| | |
|---|---|
| DHCPv4.Client.1.Enable | *true* |
| DHCPv4.Client.1.Interface | Device.IP.Interface.1 |
| | |
| DHCPv4.Client.1.SentOption.1.Enable | *true* |
| DHCPv4.Client.1.SentOption.1.Tag | 60 |
| DHCPv4.Client.1.SentOption.1.Value | “ACME Widget” (as hexBinary) |

### IV.9.2 DHCP Server Configuration (gateway)

The gateway is also relatively simple. Its downstream IP interface is IP.Interface.1.

A DHCP Server object is created and configured as follows:

| | |
|---|---|
| DHCPv4.Server.Enable | *true* |
| DHCPv4.Relay.Enable | *true* |
| | |
| DHCPv4.Relay.Forwarding.1.Enable | *true* |
| DHCPv4.Relay.Forwarding.1.Interface | Device.IP.Interface.1 |
| DHCPv4.Relay.Forwarding.1.VendorClassID | “ACME” |
| DHCPv4.Relay.Forwarding.1.VendorClassIDMode | “Prefix” |
| DHCPv4.Relay.Forwarding.1.LocallyServed | *false* |
| DHCPv4.Relay.Forwarding.1.DHCPServerIPAddress | 1.2.3.4 |
| | |
| DHCPv4.Server.Pool.1.Enable | *true* |
| DHCPv4.Server.Pool.1.Interface | Device.IP.Interface.1 |
| DHCPv4.Server.Pool.1.MinAddress | 192.168.1.64 |
| DHCPv4.Server.Pool.1.MaxAddress | 192.168.1.254 |
| DHCPv4.Server.Pool.1.ReservedAddresses | 192.168.1.128, 192.168.1.129 |
| DHCPv4.Server.Pool.1.SubnetMask | 255.255.255.0 |

If a DHCP request includes an option 60 value that begins with “ACME”, the request is forwarded to the DHCP server at 1.2.3.4. All other requests are served locally from the pool 192.168.1.64 - 192.168.1.254 (excluding 192.168.1.128 and 192.168.1.129).

## IV.10 Reconfigure an Existing Interface

The Controller might want to reconfigure an existing Interface to provide alternate routing functionality. For the purposes of this illustration, an existing Ethernet Interface that is configured for the downstream-side will be reconfigured as an upstream Ethernet Interface replacing an existing DSL Interface.

The current configuration on the upstream side looks like:
IP.Interface.1 → Ethernet.Link.1 → ATM.Link.1 → DSL.Channel.1 → DSL.Line.1

The current configuration on the downstream side contains:

- IP.Interface.2 → Ethernet.Link.2 → Bridging.Bridge.1.Port.1 (ManagementPort=true)
- Bridging.Bridge.1.Port.1 LowerLayers parameter has two references:
  - Bridging.Bridge.1.Port.2
  - Bridging.Bridge.1.Port.3
- Bridging.Bridge.1.Port.2 LowerLayers parameter has a reference of Ethernet.Interface.1
- Bridging.Bridge.1.Port.3 LowerLayers parameter has a reference of Ethernet.Interface.2

The Controller would follow these steps to reconfigure the Ethernet.Interface:

1. Determine which Ethernet.Interface is to be reconfigured. For the purpose of this illustration we will use Ethernet.Interface.1.
2. Retrieve the InterfaceStack.
3. Find the higher-layer Interface of Ethernet.Interface.1 by finding the InterfaceStack entry that has Ethernet.Interface.1 as the LowerLayer. The HigherLayer parameter of the identified InterfaceStack instance will be the Interface we are interested in, for the purpose of this illustration we found Bridging.Bridge.1.Port.2.
4. Remove the Bridging.Bridge.1.Port.2. This removal will automatically clean up the InterfaceStack instances that connect Bridging.Bridge.1.Port.1 → Bridging.Bridge.1.Port.2 and Bridging.Bridge.1.Port.2 → Ethernet.Interface.1. Also, it will remove Bridging.Bridge.1.Port.2 from the LowerLayers parameter contained within Bridging.Bridge.1.Port.1.
5. Find the DSL.Line reference within the LowerLayer parameter of the InterfaceStack.
6. Follow the InterfaceStack up to the Ethernet.Link reference by looking at the HigherLayer parameter in the current InterfaceStack instance and then finding the InterfaceStack instance containing that Interface within the LowerLayer parameter until the HigherLayer reference is the Ethernet.Link Interface. For the purpose of this illustration, we found Ethernet.Link.1.
7. Reconfigure the LowerLayers parameter of Ethernet.Link.1 such that its value is "Device.Ethernet.Interface.1" instead of "Device.ATM.Link.1".
8. The CPE updates the InterfaceStack table and sets the Upstream parameter to true on the Ethernet.Interface.1 instance automatically.
9. Note that the Controller might also want to update other related objects, including the NAT object, the Routing.Router object, or various QoS and Bridging tables. VLANs might also need to be created.

After the CWMP Session is completed and the CPE commits the configuration, the upstream side will look like:
IP.Interface.1 → Ethernet.Link.1 → Ethernet.Interface.1

## IV.11 Backup / Restore Using Vendor Configuration Files

In certain troubleshooting scenarios, a Device that has its user configuration modified in a manner that cannot be easily restored by setting individual parameters can have the Device's user configuration restored by applying a previous user configuration to the Device. When performing a backup and restoration of configuration files, the Controller can correlate the instance number of the VendorConfigFile retrieved during backup (Upload RPC) operation with the URL of the restore (Download) operation. The following sequence diagrams depict a backup and restoration scenario that correlates these attributes of a configuration file.

Figure 34 depicts a message sequence scenario where a configuration is **backed up** from the Device to the ACS using CWMP.

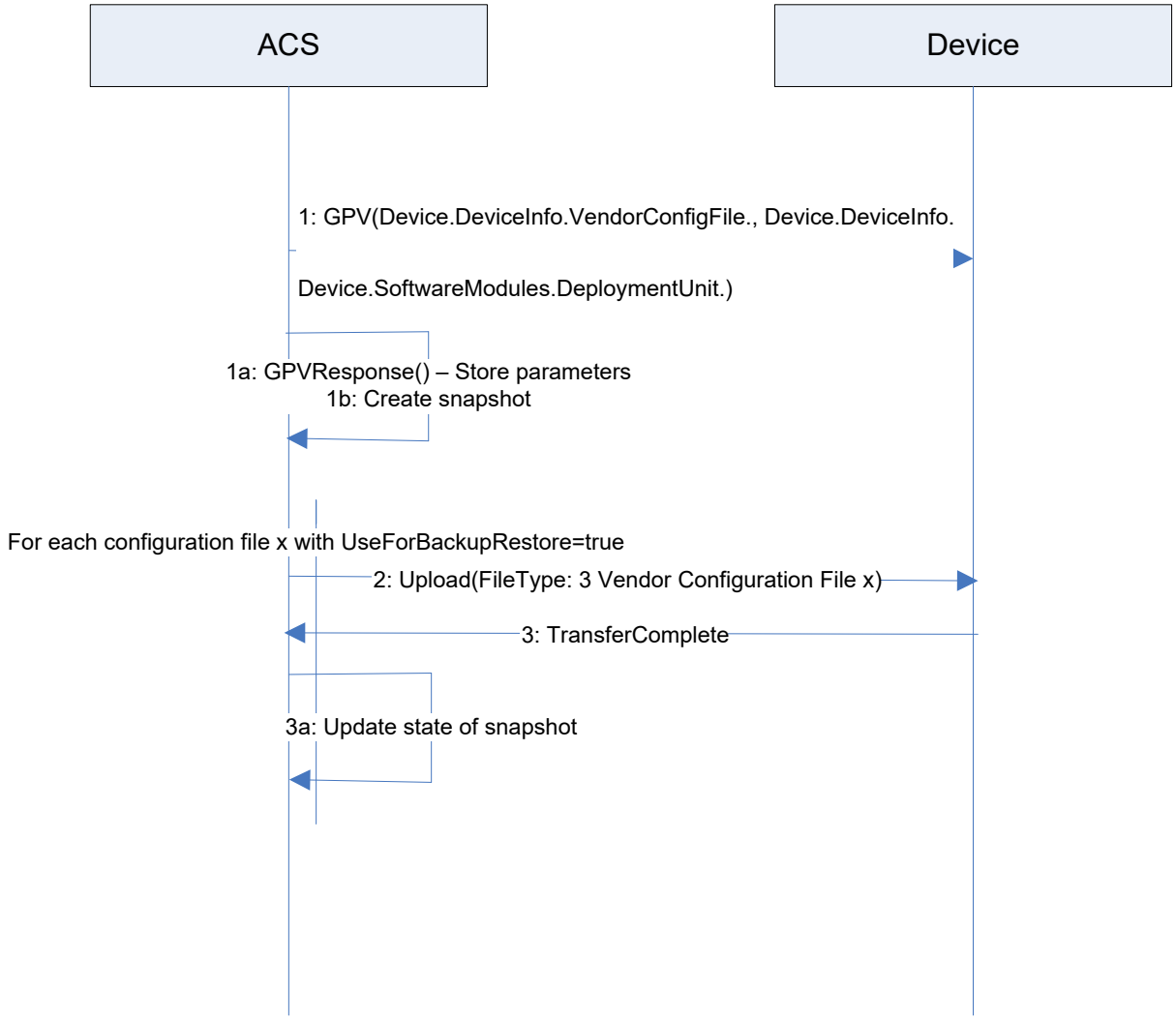



**Figure 34 - Device User Configuration Backup**

Step 1: Retrieve instances and values of VendorConfigFile and DeviceInfo:
The parameter values of the DeviceInfo and VendorConfigFile provide the information necessary to restore a Device to a point in time. Minimally the information needed to create a snapshot includes:

- Device.DeviceInfo.ManufacturerOUI
- Device.DeviceInfo.ProductClass
- Device.DeviceInfo.SerialNumber
- Device.DeviceInfo.HardwareVersion
- Device.DeviceInfo.SoftwareVersion
- Device.DeviceInfo.VendorConfigFile.{i}. (Entire object)
- Device.SoftwareModules.DeploymentUnit.{i}.UUID
- Device.SoftwareModules.DeploymentUnit.{i}.Alias
- Device.SoftwareModules.DeploymentUnit.{i}.Name
- Device.SoftwareModules.DeploymentUnit.{i}.Version
- Device.SoftwareModules.DeploymentUnit.{i}.VendorConfigList

*NOTE – Only instances of DeploymentUnit with VendorConfigFile instances with the UseForBackupRestore parameter set to the value true as items in the instance's VendorConfigList parameter will need to be backed up.*

This information is necessary as restoring Device configurations with different hardware versions, software versions or deployment units that existed at the time of the backup can result in a failed restoration attempt or a corrupted Device.

Step 1a: The parameters returned by the Device in the GetParameterValuesResponse are used to create a "snapshot" of the Device. The definition of what is needed to create a snapshot and how a snapshot is administered in an ACS is implementation specific.

Step 2: Backup each configuration file defined by the Device in the VendorConfigFile table with the UseForBackupRestore parameter set to the value "true" using the Upload RPC with a File Type "3 Vendor Configuration File x" where "x" is the instance number of the file in the VendorConfigFile table.

*NOTE – An ACS can also have additional information, outside step 1, to discern which configuration files are necessary to restore a Device, as well as the order in which the configuration files need to be restored where dependencies exist between the configuration files within the potential snapshot.*

Step 3, 3a: Upon completion of the transfer for each file via the Transfer Complete event, the ACS will update the state of the snapshot. The lifecycle and state management of the snapshot by an ACS is implementation specific.

At this point a Device snapshot exists that can be used to restore a Device to this point in time.

Figure 35 depicts a message sequence scenario where a configuration is **restored** to the Device from the ACS

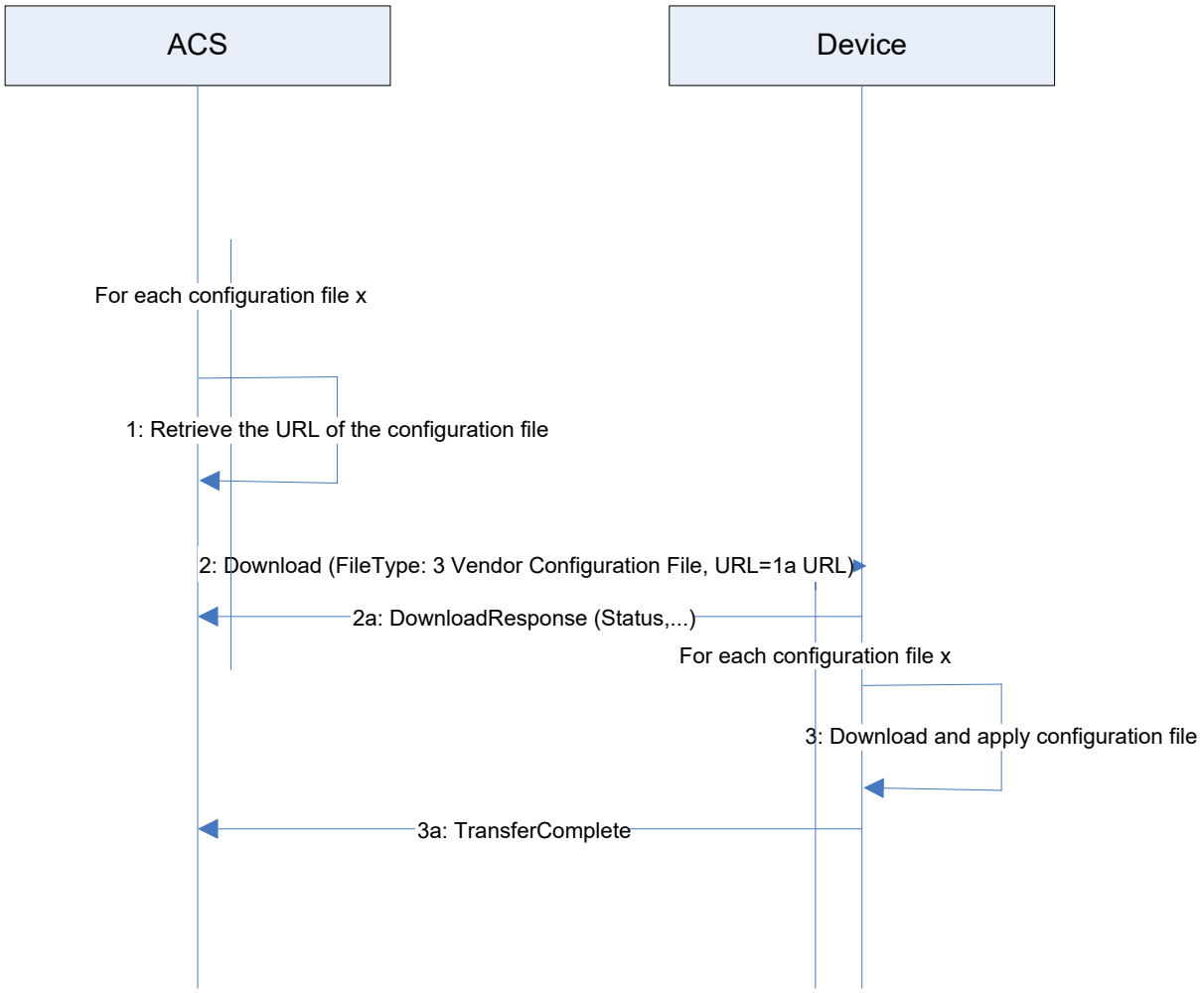



**Figure 35 - Device User Configuration Restore**

Step 1: For each user configuration file in the snapshot, retrieve the information for the location of the configuration file.

Step 2, 2a: Download the configuration using the File Type "3 Vendor Configuration File" and the location of the configuration file.

*NOTE – Other elements (e.g., credentials) might be required but are outside the scope of this sequence. When downloaded, a VendorConfigFile instance with the same value for Name or Alias (if supported and present) will update the corresponding instance in the VendorConfigFile table and will not create a new entry within the table.*

Step 3, 3a: The Device performs the download of each configuration file and responds with a Transfer Complete event.

# Appendix V IPv6 Data Modeling Theory of Operation

The Device:2 data model supports IPv6[6] via various IPv6-specific objects and parameters that are designed to be used with other IP version neutral and IPv4-specific objects and parameters. This Appendix briefly reviews all the relevant objects and parameters, and then presents some example configurations.

## V.1 IPv6 Overview

The IETF published RFC 2460 [16], Internet Protocol, *Version 6 (IPv6) Specification* in 1998. Since then, it has published a variety of RFCs to create a suite of protocols (and extensions to protocols) for operating, managing, and configuring IPv6 networks and devices. In addition there are RFCs that document transition mechanisms (to transition from IPv4 to IPv6) and best current practices (that describe which RFCs to implement depending on what a device is or needs to do).

The Broadband Forum has published several Technical Reports describing IPv6 architectures and device requirements. Specifically, TR-124 Issue 2 [30] includes IPv6 requirements for Residential Gateways (RGs), TR-177 [31] describes migration to IPv6 in the context of TR-101 [29], and TR-187 [32] describes an architecture for IPv6 for PPP Broadband Access. The Device:2 IPv6 Data Model is intended to ensure that TR-069 [2] or USP [67] managed End Devices, RGs, and other Network Infrastructure Devices can be managed and configured, consistent with the requirements listed in these documents.

The basic elements of IPv6 data modeling involve information on IPv6 capabilities, and enabling those capabilities on devices and device interfaces (see Section V.3), configuring addresses, prefixes , and configuration protocols on upstream and downstream interfaces (see Sections V.4 and V.5), interacting with other devices on the Local Area Network (LAN) (see Section V.6), and configuring IPv6 routing and forwarding information (see Section V.7).

Configuration protocols include Neighbor Discovery (ND; RFC 4861 [22]) and DHCPv6 (RFC 3315 [18]). Neighbor Discovery includes several messages that are important to configuration, including Router Solicitation (RS) [sent by devices looking for routers], Router Advertisement (RA) [sent by routers to other devices on the LAN], Neighbor Solicitation (NS) [used to identify if any other device on the LAN is using the same IPv6 address, and used to see if previously detected devices are still present; the latter is called Neighbor Unreachability Detection (NUD)], and Neighbor Advertisement (NA) [used to respond to a NS sent to one of the device's IPv6 addresses]. These messages are central to the stateless address autoconfiguration (SLAAC) mechanism described in RFC 4862 [23]. SLAAC is expected to be the primary means of IPv6 address configuration for devices inside a home network. RFC 4191 [20] extended the RA message to support a RouteInformation option. RFC 6106 [26] extended the RA message to support sending Recursive DNS Servers (RDNSS) information for DNS configuration.

[6] *Introduced in Amendment 2*

DHCPv6 can also be used for IPv6 address provisioning, through its IA_NA option. DHCPv6 was extended by RFC 3633 [19] to provide the IA_PD option for delegating IPv6 prefixes to routers (that the routers can then use to provide IPv6 addresses to other devices on the LAN, or to further sub-delegate to other routers inside the LAN). Both IA_NA and IA_PD require the DHCPv6 server to maintain state for these assignments (since they have lifetimes, can expire, and require renewal). DHCPv6 can also supply a variety of stateless configuration options, including recursive DNS server information. RGs can have both DHCPv6 client and server, and it may be desirable for some of the stateless options to be passed through from the client to the server.

Interfaces that support IPv6 will have more than one IPv6 address. IPv6 interfaces are always required to have a link-local address (described in RFC 4862 [23]). Other IPv6 addresses may be acquired through SLAAC, DHCPv6 IA_NA, or they may be statically configured. Routers may acquire prefixes (for use with address assignment in the LAN) from DHCPv6 IA_PD, static configuration, or by generating their own Unique Local Address (ULA) prefixes from a self-generated ULA Global ID (RFC 4193 [21]).

Because of the various IPv6 addresses that devices can have, maintaining good routing table and IPv6 forwarding information is critical. Route information can be obtained from received RA messages (both by noting that the sending device is a router, and from the RouteInformation option) as well as other protocols.

## V.2 Data Model Overview

This Theory of Operations focuses on data modeling for the purpose of establishing upstream and downstream connectivity for TR-069 [2] or USP [67] enabled devices, and for configuration of IPv6-related parameters. This is not an exhaustive description of data model changes made in support of IPv6, and only intends to describe the working of elements that are not readily obvious.

The following tables are key to IPv6 data modeling:

- *IP*
  - *IP.Interface*
    - *IP.Interface.IPv6Address*
    - *IP.Interface.IPv6Prefix*
- *PPP.Interface*
- *Routing.Router*
  - *Routing.Router.IPv6Forwarding*
  - *Routing.RouteInformation.InterfaceSetting*
- *NeighborDiscovery.InterfaceSetting*
- *RouterAdvertisement.InterfaceSetting*
  - *RouterAdvertisement.InterfaceSetting.Option*
- *Hosts.Host*

- *DHCPv6*
  - *DHCPv6.Client*
    - *DHCPv6.Client.Server*
    - *DHCPv6.Client.SentOption*
    - *DHCPv6.Client.ReceivedOption*
  - *DHCPv6.Server*
    - *DHCPv6.Server.Pool*
      - *DHCPv6.Server.Pool.Client*
        - *DHCPv6.Server.Pool.Client.IPv6Address*
        - *DHCPv6.Server.Pool.Client.IPv6Prefix*
        - *DHCPv6.Server.Pool.Client.Option*
      - *DHCPv6.Server.Pool.Option*

Note that the following tables have separate theories of operation, and are not described again here:

- *IPv6rd.InterfaceSetting*
- *DSLite.InterfaceSetting*

*Firewall* includes some IPv6 elements that are not described, since it does not interact with tables other than an association with *IP.Interface*. As such, its IPv6 usage is considered straightforward, and explanation is considered unnecessary.

Similarly, *DNS.Client.Server* is not described.

Use of DHCPv6 elements of *Bridging.Filter* are also not described, as there is no conceptual difference between how they are used and how DHCPv4 elements are used.

Figure 36 shows the relationship of IPv6 configuration messages to devices and the tables used to configure the protocol messages and store the responses.

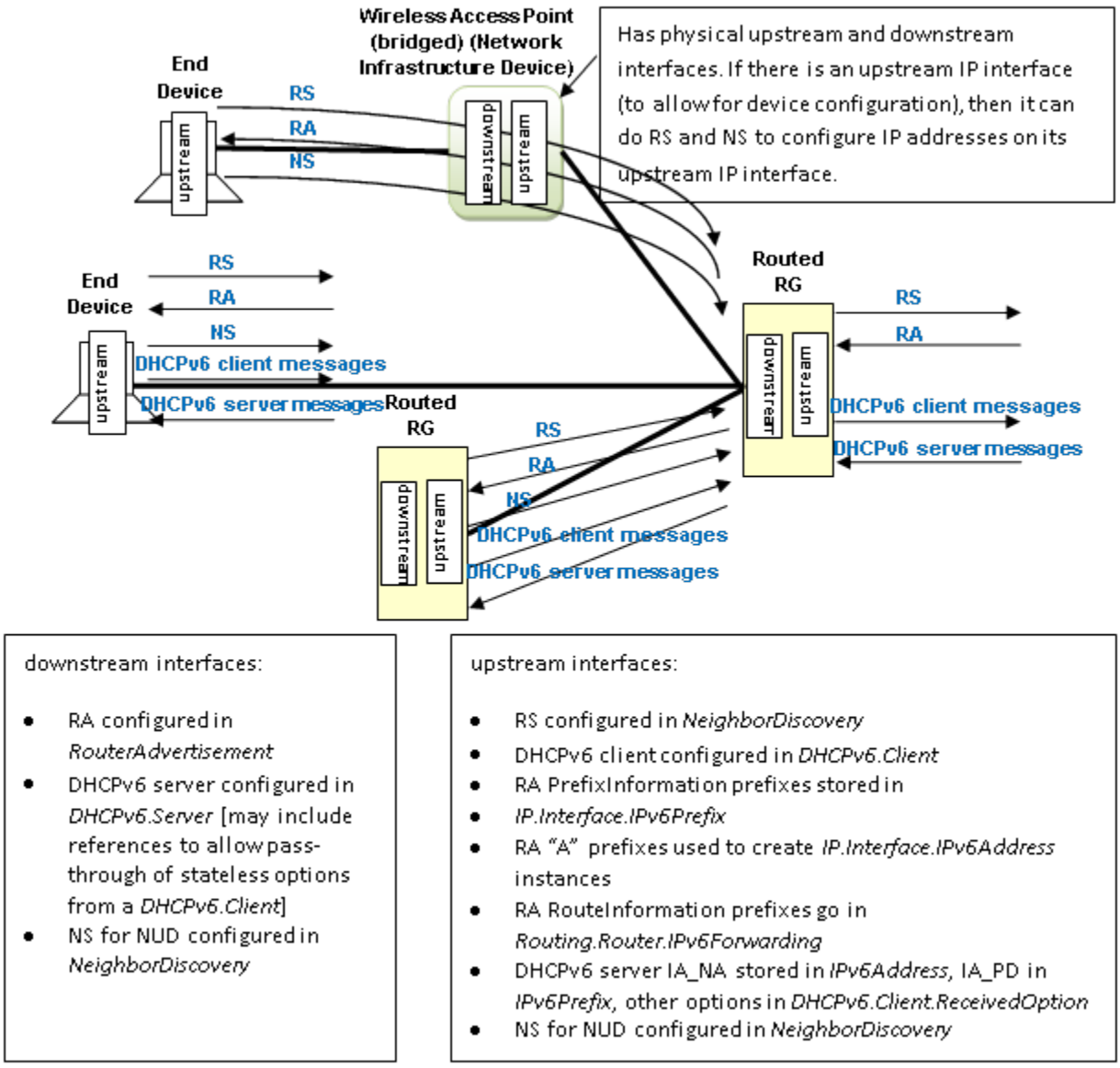



**Figure 36 – Relationship of Protocols to Data Model**

Figure 37 shows internal relationships of parts of the data model involved in IPv6 addresses and IPv6 prefixes. The following sections describe in greater detail how these various tables are populated.

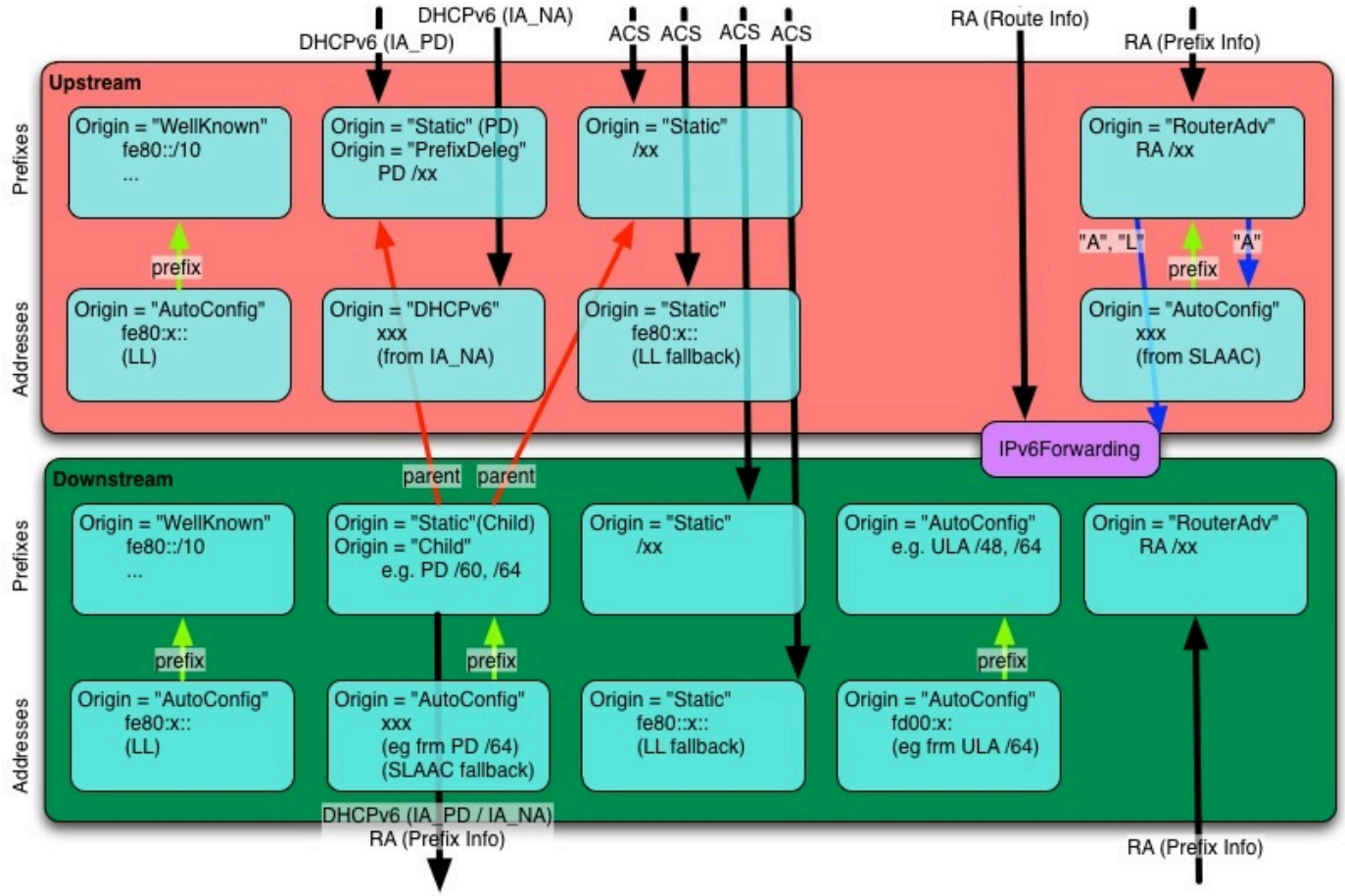



**Figure 37 – Internal Relationships of IPv6 Addresses and Prefixes**

## V.3 Enabling IPv6

The *IP IPv6Capable* parameter indicates whether the device supports IPv6. *IP.IPv6Enable* controls enabling IPv6 is on the device. IPv6 can only be enabled on a device with *IPv6Capable=true*. *IPv6Status* indicates whether IPv6 has been enabled on the device.

Per TR-124 Issue 2 [30], the upstream interface can be configured to establish an IPv6 connection either over PPP (PPPoA or PPPoE) or directly over Ethernet. Both mechanisms require an *IP.Interface* instance with *IPv6Enable* set to *true*. When using PPP, a *PPP.Interface* instance must have *IPv6CPEnable* set to *true* (which can only occur if *PPP.SupportedNCPs* includes *IPv6CP* in its list of Network Control Protocols (NCPs)).

Enabling IPv6 on specific downstream or upstream interfaces requires that *IP.Interface* instances have *IPv6Enable* set to *true*.

## V.4 Configuring Upstream IP Interfaces

An upstream IP Interface is an *IP.Interface* that is associated with an *Upstream=true* physical interface, via the *InterfaceStack*. Every *Upstream=true* physical interface that will be used to support routed IPv6 traffic will have an upstream IP Interface for each distinct upstream IPv6 connection that is established over that physical interface.

Upstream IPv6 connections can be established on an upstream IP Interface either through internal logic (for well-known addresses and the link-local address), static configuration, or dynamically through received Router Advertisement (RA) messages or DHCPv6 client behaviors. Received RA and DHCPv6 messages can contain configuration information for more than just establishing the upstream IP interface. The data model allows for the storage of additional configuration information sent by one of these protocols.

### V.4.1 Configuration Messages Sent Out the Upstream IP Interface

The device can be configured to send Router Solicitation and DHCPv6 client messages out an upstream IP interface.

- A device that is configured to send Router Solicitation messages out an upstream IP interface will have a *NeighborDiscovery.InterfaceSetting* instance whose *Interface* is the related upstream *IP.Interface*, and with *RSEnable=true*.

- A device that is configured to send DHCPv6 client requests out an upstream IP interface will have a *DHCPv6.Client* instance whose *Interface* is the related upstream *IP.Interface*, and with *Enable=true*. *RequestAddresses* indicates whether IA_NA is to be requested, *RequestPrefixes* indicates whether IA_PD is to be requested, and *RequestedOptions* identifies which other options are to be requested. *DHCPv6.Client.Server*, *DHCPv6.Client.SentOption*, and *DHCPv6.Client.ReceivedOption* are populated as appropriate, as described in the data model.

### V.4.2 IPv6 Prefixes

*IP.Interface.IPv6Prefix* instances on upstream IP interfaces are used to store all prefixes received in RA messages on the interface (with *Origin* of *RouterAdvertisement*), prefixes delegated by DHCPv6 IA_PD (with *Origin* of *PrefixDelegation*), statically configured IPv6 prefixes (but only the ones that are intended to be sub-divided for use on downstream interfaces with sent RA messages or DHCPv6 server functions), and *WellKnown* prefixes, as appropriate (such as certain well-known multicast prefixes, where the device joins the multicast group for that prefix on that interface).

*RouterAdvertisement* prefixes with *Autonomous=true* are used to create an *IPv6Address* instance on the interface, and can be used to create routes in *Routing.Router.IPv6Forwarding.* *RouterAdvertisement* prefixes with *OnLink=true* can also be used to create routes in

*Routing.Router.IPv6Forwarding.* Prefixes received in a RA RouteInformation option are not stored with the interface, but rather in an instance of *Routing.RouteInformation.InterfaceSetting.*

*PrefixDelegation* prefixes and *Static* prefixes are not directly used on the upstream IP interface. They are prefixes that are intended to be sub-divided for use on the device's downstream interfaces, either by the DHCPv6 server for IA_NA or IA_PD, sent in RA messages (as on-link and/or autonomous prefixes), or used to self-assign addresses to other interfaces on the device. Non IA_PD prefixes received in DHCPv6 options are not stored with the upstream IP interface. Prefixes for static routes are entered directly into *Routing.Router.IPv6Forwarding* and do not need to also have upstream IP interface *IPv6Prefix* entries.

It is often desirable to configure information about delegated prefixes before they have been delegated (for example, that a particular /64 of that prefix is to be used on the downstream interface for address assignment). In order to allow for the referencing of not-yet-existing-but-expected delegated prefixes, an *Origin=Static IPv6Prefix* entry is created of *Type=PrefixDelegation*. When a device receives a delegated prefix, it is expected to first look for such Static entries and populate them with the delegated prefix information, instead of creating a new *IPv6Prefix* instance of *Origin=PrefixDelegation*. How these references are configured on downstream interfaces is discussed in Section V.5.1.

### V.4.3 IPv6 Addresses

IPv6 link-local addresses on an upstream IP Interface are generally internally generated, although they can be configured statically, when necessary (when the internal default link-local address fails Duplicate Address Detection (DAD)). A properly configured upstream *IP.Interface* instance will have a *IP.Interface.IPv6Address* instance for its link-local address. This will have *Origin* of *AutoConfigured* (if internally generated per RFC 4862 [23]) or *Static* (if statically configured by some management entity).

IPv6 addresses that are created via stateless address autoconfiguration (SLAAC), as defined in RFC 4862 (from received RA messages that contain prefix(es) with *Autonomous=true*) cause the device to create a *IP.Interface.IPv6Address* instance with *Origin* of *AutoConfigured.* IPv6 addresses assigned via DHCPv6 IA_NA cause the device to create a *IP.Interface.IPv6Address* instance with *Origin* of *DHCPv6*. Statically created IPv6 addresses will have *Origin* of *Static*. If any of these addresses are Global Unicast Addresses (GUA), they can be used to originate and terminate traffic to/from either the downstream or the upstream, independent of which physical interface they are associated with.

## V.5 Configuring Downstream IP Interfaces

A downstream IP Interface is a *IP.Interface* that is associated with an *Upstream=false* physical interface, via the *InterfaceStack*. As noted in the definition of the *Upstream* parameter, "For an End Device, *Upstream* will be *true* for all interfaces." This means that only RGs or (possibly) other Network Infrastructure Devices will have downstream IP Interfaces.

### V.5.1 IPv6 Prefixes

*IP.Interface.IPv6Prefix* instances on downstream IP interfaces are used to store all prefixes that are either on-link for that downstream IP interface, or can be delegated to or used by routers connected to that downstream IP interface. On-link prefixes include prefixes that are included in Router Advertisement (RA) messages for SLAAC (Autonomous prefixes), those used as DHCPv6 address pools, and those used for static addressing by End Devices that connect to that downstream IP interface.

The device can have a Unique Local Address (ULA) /48 prefix defined in *IP.ULAPrefix*. In general, the device will generate its own ULA /48 prefix, although this value could be configured directly by the user or through TR-069 [2] or USP [67]. If ULA addressing is to be supported on a downstream interface, then *IP.Interface.ULAEnable* must be *true*. The ULA /48 prefix can be associated with any downstream IP interface, and can be sub-divided to provide ULA prefixes on multiple downstream IP interfaces (by assigning longer prefixes from the ULA /48 prefix to these downstream IP interfaces). When the device creates a ULA prefix on a downstream interface, it creates an *IPv6Prefix* instance with *Origin=AutoConfigured*.

RGs that are configured to act as routers need to know which prefixes to include in their sent Router Advertisement (RA) messages and to be used in DHCPv6 server pools. These prefixes need to be associated with the downstream IP interface for those *RouterAdvertisement.InterfaceSetting* and *DHCPv6.Server.Pool* instances. These prefixes can be statically configured on the downstream IP interface, or they can be automatically generated from prefixes on an upstream IP interface with *Origin* of *PrefixDelegation* or *Static*, or they can be generated from the ULA /48 prefix (as described in the previous paragraph). Prefixes that are automatically (by internal code) derived from prefixes on an upstream IP interface with *Origin* of *PrefixDelegation* or *Static*, will point to that upstream IP interface in *ParentPrefix* and have *Origin* of *Child*.

It is often desirable to pre-configure information about prefixes on a downstream IP interface that are to be derived from delegated (on the upstream interface) prefixes. This will need to be done before that prefix has been delegated and without knowledge of what that prefix will be. A derived-from-not-yet-existing-but-expected-delegated-prefix downstream IP interface *IPv6Prefix* entry will have *Origin=Static* and *Type=Child*, and will have *ParentPrefix* pointing to an upstream IP interface *IPv6Prefix* instance (that is *Origin=Static* and *Type= PrefixDelegation*). When a device receives a delegated prefix and populates the upstream IP interface IPv6Prefix instance, and needs to generate downstream IP interface prefixes from that delegated prefix, it is expected to first look for such *Static Child* entries and populate them with the derived prefix information, instead of creating a new *IPv6Prefix* instance of *Origin=Child*. How the referenced parent prefixes are configured on upstream IP interfaces is discussed in Section V.4.2.

If the device receives RA messages on downstream IP interfaces, autonomous and on-link prefixes in such received RA message Prefix Information options can also be recorded in *IP.Interface.IPv6Prefix*. At this time, there is no additional guidance for using the information in these RA messages received on downstream interfaces. They are simply stored, to provide information about other devices in the home network.

### V.5.2 IPv6 Addresses

As with the upstream IP interfaces, IPv6 link-local addresses on a downstream IP interface are generally internally generated, although they can be configured statically, when necessary (when the internal default link-local address fails Duplicate Address Detection (DAD)). A properly configured downstream IPv6 connection will have a *IP.Interface* instance with a *IP.Interface.IPv6Address* instance for its link-local address. This will have *Origin* of *AutoConfigured* (if internally generated per RFC 4862 [23]) or *Static* (if statically configured by some management entity).

If the device has a Unique Local Address (ULA) prefix that it is advertising and/or sub-delegating to devices on the LAN, then it needs to have at least one address from this prefix assigned to downstream IP interfaces that expect to support usage of the ULA.

If the device did not receive an address on its upstream IP interface (from DHCPv6 or SLAAC), but it was delegated a prefix (DHCPv6 IA_PD), then it is expected to assign an address from a prefix (*Origin=Child* or *Type=Child*) derived from that delegated prefix to one of its non-upstream interfaces. This *IPv6Address* instance will have *Origin* of *AutoConfigured*. This address can be used for originating and terminating messages to and from either the downstream or the upstream interfaces.

## V.6 Device Interactions

The RG can interact with other devices on the LAN both by actively sending messages with or without configuration information, and by passively listening to messages received from other devices. End Devices can interact with other devices on the LAN by passively listening to messages received from other devices and by actively performing Neighbor Unreachability Detection (NUD) to determine if previously detected devices are still reachable.

### V.6.1 Active Configuration

To assist in the automated configuration of other devices on the LAN, an RG sends Router Advertisement (RA) messages and DHCPv6 server messages. This function is associated with downstream IP interfaces, and thus does not apply to End Devices. As noted in the above section on downstream IP interfaces, only RGs or other infrastructure devices will have downstream IP interfaces.

- *RouterAdvertisement.InterfaceSetting* instances whose *Interface* is the related downstream *IP.Interface*, with *Enable=true*, define the content of RA messages that get sent on the downstream IP interface. The *RouterAdvertisement.InterfaceSetting* instance will include references to *IPv6Prefix* entries in the associated downstream IP interface. These are *IPv6Prefix* entries of *Origin=Child* or *Origin=Static*.

- *DHCPv6.Server.Pool* instances whose *Interface* is the related downstream *IP.Interface*, with *Enable=true*, contain information for filtering DHCPv6 client requests, and identify the IPv6 prefix(es) (references to *IPv6Prefix* entries of the associated downstream IP interface) that provide the pool of IPv6 addresses and IPv6 prefixes available for assignment from this pool. Information on soliciting clients (including assigned addresses and prefixes and received option information) is stored in *DHCPv6.Server.Pool*.*Client*. Additional options that are sent to soliciting clients is stored in *DHCPv6.Server.Pool*.*Option*. The *PassthroughClient* parameter in this table identifies whether the value of this option is simply passed through from a DHCPv6 client on an upstream interface.

As noted above, both *RouterAdvertisement.InterfaceSetting* and *DHCPv6.Server.Pool* have references to *IPv6Prefix* entries. The *ManualPrefixes*, *IANAManualPrefixes* and *IAPDManualPrefixes* parameters allow for configuration (through TR-069 [2],USP [67], user interface, or other means) of prefixes that are to be included in RA messages, and to be used in deriving DHCPv6 IA_NA and IA_PD offers, respectively. The *Prefixes*, *IANAPrefixes*, and *IAPDPrefixes* parameters list all of the prefixes that the devices actually does include in these messages. Since the **ManualPrefixes* entries may point to *IPv6Prefix* entries that are not enabled, it is possible that not all of those will be included in these parameters' lists. In addition to the **ManualPrefix* entries, these lists may also include references to prefixes that the device creates or uses automatically in RA messages or for deriving DHCPv6 IA_NA or IA_PD offers.

There is some flexibility in the modeling of ULA IA_PD prefixes. It is not required to model the ULA /48 prefix in an *IPv6Prefix* instance. If the ULA /48 is not represented in an IPv6Prefix instance and *ULAEnable* is *true* for a downstream interface and *IAPDEnable* is *true* for a *DHCPv6.Server.Pool* instance, then it can be assumed that the device will sub-delegate prefixes from the ULA /48 prefix. Alternately, the ULA /48 can be included as an *AutoConfigured* prefix in a downstream interface, and that *IPv6Prefix* instance can be referenced in *IAPDPrefixes* in the *DHCPv6.Server.Pool* instance. It is also possible to manually create a *Static* longer-than-/48 prefix from the ULA prefix in a downstream interface. This *Static* prefix can then be referenced in *IAPDManualPrefix* for a *DHCPv6.Server.Pool* instance for that interface.

For IA_PD, there is one additional parameter: *IAPDAddLength*. This parameter is configured to recommend how many bits should be added to an *IAPDPrefixes* prefix to create a delegated prefix offer.

### V.6.2 Monitoring

All devices can monitor and record information from messages sent by other devices.

- Information received in Neighbor Solicitation (NS) and Neighbor Advertisement (NA) messages sent by other devices is recorded in *Hosts.Host.*
- In order to actively solicit information from other devices on the LAN, the device can have a *NeighborDiscovery.InterfaceSetting* instance whose *Interface* is the related downstream *IP.Interface*, and with *NUDEnable=true*. To determine whether there are other routers connected to the LAN that are behaving like IPv6 routers to this same LAN segment, this *InterfaceSetting* can also have *RSEnable=true*. However, it is not recommended that routers do this until there is better guidance available for routers that co-exist in a peered environment on the same LAN.

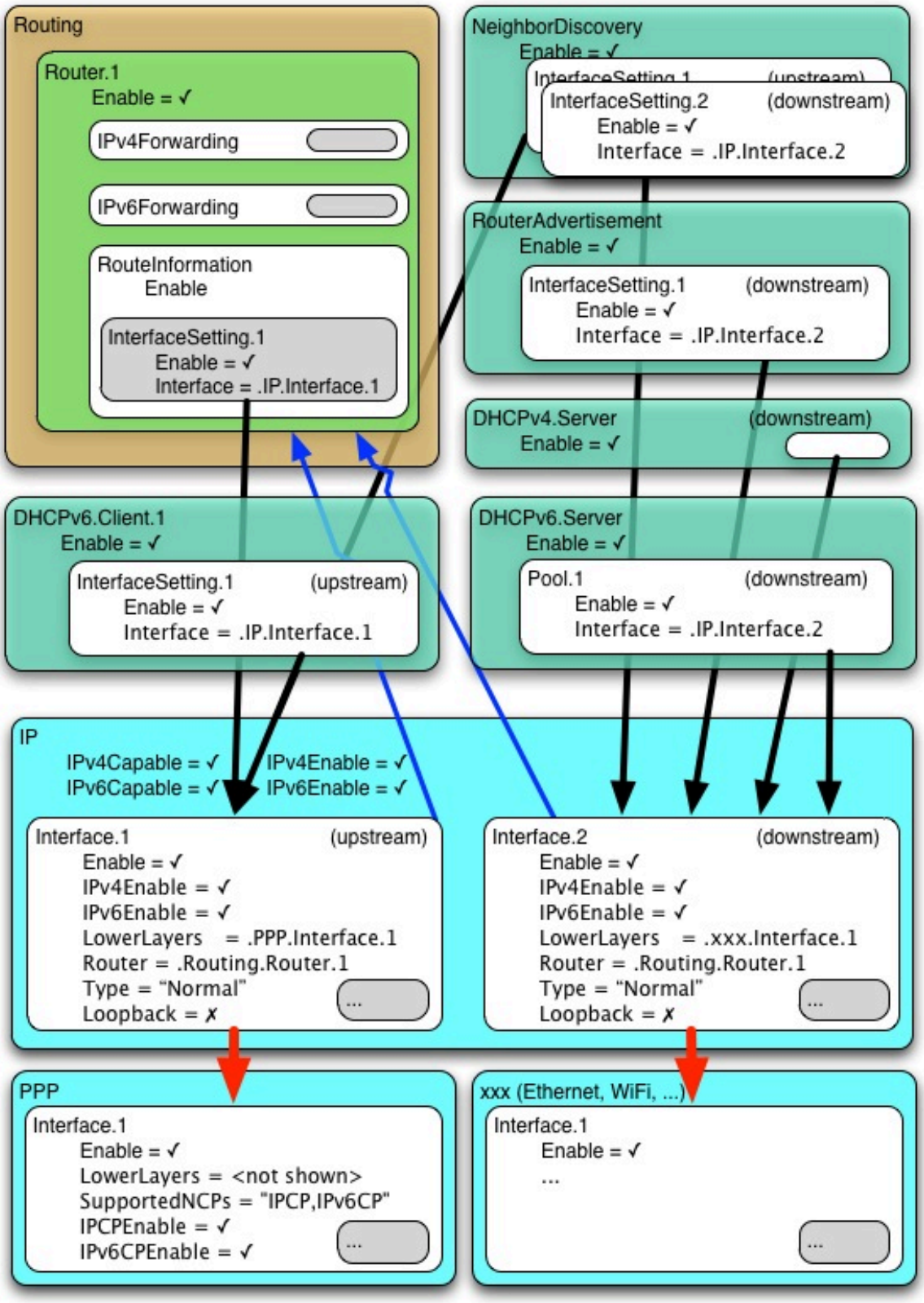
## V.7 Configuring IPv6 Routing and Forwarding

IPv6 routing information is stored in instances of *Routing.Router.IPv6Forwarding.* This information can in part be derived from Router Advertisement (RA) messages, either directly from the address of the router sending the RA, or from RA RouteInformation (RFC 4191 [20]) options that may be included in the message. *Routing.RouteInformation.InterfaceSetting* instances record received RA RouteInformation options.

## V.8 Configuring IPv6 Routing and Forwarding

Following is an example of how a typical RG (one upstream and one downstream interface, with delegated prefix and IA_NA address, and ULA enabled) might be configured. The corresponding data model is shown below the figure. Not all parameters are shown, and objects and parameters that the Controller is likely to have explicitly created or written are shown in **bold face** (some of these settings might alternatively be present in the factory default configuration).




```
# IP
IP.
```

```
      IPv6Capable = true
      IPv6Enable = true
      IPv6Status = "Enabled"
      ULAPrefix = fd01:2345:6789::/48     # typically generated by CPE

# Router Solicitation (Upstream IP interface)
NeighborDiscovery.
      Enable = true
      InterfaceSetting.1.
            Enable = true
            Interface = IP.Interface.1
            RSEnable = true

# DHCPv6 Client (Upstream IP interface)
DHCPv6.Client.1
      Enable = true
      Interface = IP.Interface.1
      RequestAddresses = true
      RequestPrefixes = true

# Upstream IP interface
# - Assumes DHCPv6 IA_PD will be 1080:0:0:800::/56 (this is NOT known at
#   configuration time).
# - Assumes RA(PI) will be 2001:0DB8::/64 (this is NOT known at configuration
#   time)
# - Assumes link-layer address is 55:44:33:22:11:00
#   [Section 4/RFC 2464[17]],[Section 4.1/RFC 5072[24]]
IP.Interface.1
      Enable = true
      IPv6Enable = true

      # Upstream IP interface IPv6 prefixes
      # - Assumes that the WellKnown Link Local fe80::/10 prefix not modeled
      IPv6Prefix.1
            Enable = true
            Prefix = 1080:0:0:800::/56    # DHCPv6(IA_PD) [RFC 3633[19]]
            Origin = "Static"
            StaticType = "PrefixDelegation"

      # Upstream IP interface IPv6 addresses (LL, GUA)
      IPv6Address.1
            Enable = true
            IPAddress = fe80::5544:33ff:fe22:1100
            Origin = "AutoConfigured"           # LL
            Prefix = ""
      IPv6Address.2
            Enable = true
            IPAddress = 1080:0:0:700::
            Origin = "DHCPv6"             # GUA (from IA_NA [RFC 3315[18]])
            Prefix = ""

# Downstream IP interface
# - Assumes link-layer address is 00:11:22:33:44:55 [Section 4/RFC 2464[17]]
IP.Interface.2
      Enable = true
      IPv6Enable = true
      ULAEnable = true
```

```
      # Downstream IP interface IPv6 prefixes
      IPv6Prefix.1
            Enable = true
            Prefix = 1080:0:0:800::/64
            Origin = "Static"
            StaticType = "Child"          # IA_PD /64 (for lcl, RA and IA_NA)
            ParentPrefix = IP.Interface.1.IPv6Prefix.1
            ChildPrefixBits = 0:0:0:00::/64
      IPv6Prefix.2
            Enable = true
            Prefix = 1080:0:0:810::/60
            Origin = "Static"
            StaticType = "Child"          # IA_PD /60 (for IA_PD)
            ParentPrefix = IP.Interface.1.IPv6Prefix.1
            ChildPrefixBits = 0:0:0:10::/60
      IPv6Prefix.3
            Enable = true
            Prefix = fd01:2345:6789::/48
            Origin = "AutoConfigured"     # ULA /48
      IPv6Prefix.4
            Enable = true
            Prefix = fd01:2345:6789:0::/64
            Origin = "AutoConfigured"     # ULA /64 (for lcl, RA and IA_NA)
      IPv6Prefix.5
            Enable = true
            Prefix = 2001:0db9::/60       # RA(PI) [RFC 4861[22]]
            Origin = "RouterAdvertisement"      # from peer router
            Autonomous = true
            OnLink = true

      # Downstream IP interface IPv6 addresses (LL, GUA?, ULA)
      IPv6Address.1
            Enable = true
            IPAddress = fe80::0011:22ff:fe33:4455
            Origin = "AutoConfigured"     # LL
            Prefix = ""
      IPv6Address.2
            Enable = false                # have upstream GUA so disabled
            IPAddress = 1080:0:0:800::
            Origin = "AutoConfigured"     # GUA (from IA_PD /64)
            Prefix = IP.Interface.2.IPv6Prefix.1
      IPv6Address.3
            Enable = true
            IPAddress = fd01:2345:6789::0011:22ff:fe33:4455
            Origin = "AutoConfigured"     # ULA (from ULA /64)
            Prefix = IP.Interface.2.IPv6Prefix.4

# Router Advertisement (Downstream IP interface)
RouterAdvertisement.
      Enable = true
      InterfaceSetting.1
            Enable = true
            Interface = IP.Interface.2
            ManualPrefixes = IP.Interface.2.IPv6Prefix.2

# DHCPv6 server (Downstream IP interface)
```

```
DHCPv6.Server.
      Enable = true
      Pool.1
            Enable = true
            Interface = IP.Interface.2
            <filter criteria>
            IANAManualPrefixes = IP.Interface.2.IPv6Prefix.1
            IAPDManualPrefixes = IP.Interface.1.IPv6Prefix.1,
                                 IP.Interface.2.IPv6Prefix.2
            IAPDADDLength = 4
```

# Appendix VI 6rd Theory of Operation

See Annex B for general information on how tunneling is modeled.

## VI.1 RFC 5969 Configuration Parameters

RFC 5969 [25] describes the general operation of the 6rd protocol and configuration of external parameters needed to do the protocol. Table 13 shows the 6rd configuration parameters defined in RFC 5969 and their mapping into the Device:2 data model. Refer to RFC 5969 for further description on use of these parameters.

Note that while RFC 5969 allows for multiple Border Relay (BR) IPv4 addresses, it does not describe how a device selects from among these. The device will need to have internal logic to handle this case, but service providers might wish to ensure that they know what the behavior will be, if they intend to supply multiple BR addresses.

**Table 13 – RFC 5969 Configuration Parameter Mapping**

| RFC 5969 (Section 7) Configuration Parameter | Device:2 (IPv6rd.InterfaceSetting.{i}) Parameter |
|---|---|
| IPv4MaskLen | IPv4MaskLength |
| 6rdPrefix | SPIPv6Prefix (expressed with prefix length) |
| 6rdPrefixLen | |
| 6rdBRIPv4Address | BorderRelayIPv4Addresses |

## VI.2 Internal Configuration Parameters

*AddressSource*, *TunnelInterface*, *TunneledInterface*, and *AllTrafficToBorderRelay* parameters are used to define internal device operation. *AddressSource* allows the desired source IPv4 address to be selected (to be embedded in the 6rd IPv6 address, after removing IPv4MaskLength bits from the beginning of the address, and as the source IPv4 address of the encapsulating IPv4 header). *TunnelInterface* and *TunneledInterface* allow for internal forwarding, routing, encapsulation, classification and marking of IPv6 packets. *AllTrafficToBorderRelay* impacts determination of the IPv4 destination address of the encapsulating IPv4 header.

## VI.3 IPv4 Address Source

In general, it is expected that the device will use the IPv4 address obtained on the upstream interface as the address that is embedded in the 6rd IPv6 address, and used as the encapsulating source IPv4 address. However, there could be cases where the device has other public IPv4 addresses assigned to it, and it would be preferable to use one of these. For example, if the device has a public static IP address assigned to a different interface, it could be desired to use that address instead of the address assigned to the upstream interface.

If this parameter is not present, or if it is an empty string, the device will use internal logic to determine the source IPv4 address. In cases where there is a single upstream interface with an assigned (e.g. DHCPv4, IPCP, static) IPv4 address, that is the address that will be used.

Note that service providers need to be careful when using alternate addresses. If the alternate address does not have the same higher order IPv4 bits as other devices that will be supported by the same 6rd prefix, then the IPv4 mask will need to be zero. Masked IPv4 bits will be the same for all IPv4 addresses within a 6rd domain, per RFC 5969 [25].

## VI.4 Sending All Traffic to the Border Relay Server

The default behavior of a 6rd client device is that all IPv6 packets are encapsulated in IPv4 packets with destination address of a 6rd border relay server, ***except*** when the IPv6 destination address begins with *SPIPv6Prefix*. When the destination IPv6 address begins with *SPIPv6Prefix*, then the encapsulating IPv4 destination address is derived from the IPv6 destination address by taking the next 32 - *IPv4MaskLength* bits, pre-pending the bits that are masked (as determined by its own WAN IPv4 address), and using the resulting IPv4 address as the encapsulating destination IPv4 address.

For example, if

- the IPv6 destination address is 2001:db8:64c8:200:x:x:x:x [note 64 hex = 100 decimal, c8 hex = 200 decimal, leading zeroes between colons are not shown]
- the *SPIPv6Prefix* is 2001:db8::/32
- the device's WAN IPv4 address is 10.100.100.1
- *IPv4MaskLength* is 8
- advertised-to-LAN SLAAC prefix of 2001:db8:6464:100::/64

…then the encapsulation destination IPv4 address becomes the first 8 bits of the device's WAN IPv4 address (10 for an address of 10.100.200.2), plus the next 24 bits (32-8=24) after the *SPIPv6Prefix* (next 24 bits are 64c802 hex = 100.200.2 binary). The source encapsulating IPv4 address is 10.100.100.1. The source IPv6 address begins with the prefix 2001:db8:6464:100::/64.

However, if *AllTrafficToBorderRelay* is True, then all external-bound IPv6 traffic is sent to the border relay.

This Boolean field is reflected in the routing table. If the value is False (default behavior), then the IPv6 routing table for this example (with a border relay IPv4 address of 10.0.0.1) would include the following entries:

```
::/0 -> 6rd-tunnel-interface-int0 via 2001:db8:0:100::
     (default route to border relay)
2001:db8::/32 -> 6rd-tunnel-interface-int0
     (direct connect to 6rd tunnel interface if the first 32 bits of
     destination address match SPIPv6Prefix)
2001:db8:6464:100::/64 -> Ethernet0 (downstream interface)
```

If the *AllTrafficToBorderRelay* field is true, then the 2nd entry above does not exist

## VI.5 Internal Treatment of IPv6 Packets

Since a device can have multiple upstream and multiple downstream interfaces, the model supports a logical representation of the internal virtual 6rd IPv6 interface according to the general pattern described in Annex B.

The internal virtual 6rd IPv6 interface is modeled as (*TunnelInterface,TunneledInterface*).

The IPv6Forwarding entries (which correspond to the routing table entries mentioned above) will route traffic between the downstream IPv6 interfaces and the 6rd IPv6 interface. IPv4Forwarding entries are unaffected.

Figure 38 shows the flow of tunneled 6rd traffic through the downstream, upstream, and the logical tunnel interfaces. Noted in the figure are sample values for the various *IP.Interface* entries that would be needed.

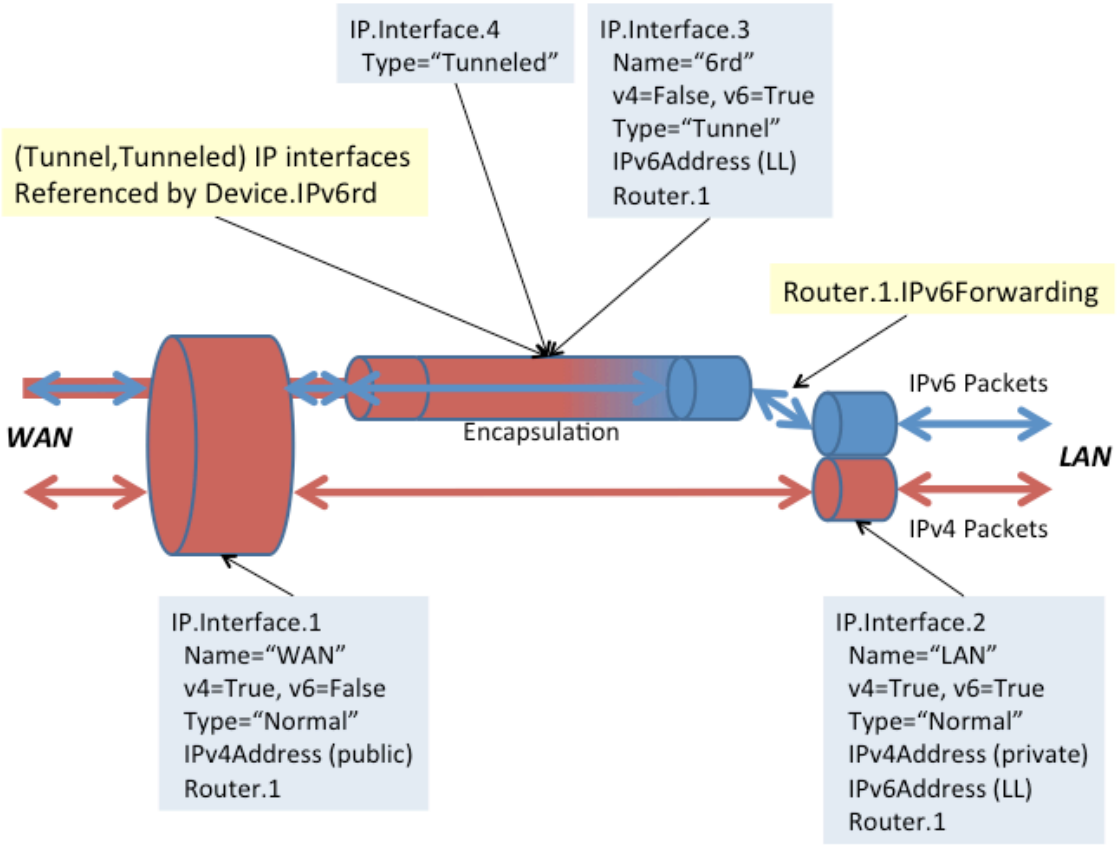



**Figure 38 – Sample 6rd Routing and Forwarding**

# Appendix VII Dual-Stack Lite Theory of Operation

See Annex B for general information on how tunneling is modeled.

RFC 6333 [27] describes the general operation of the dual-stack lite (DS-Lite) technology and configuration of external parameters needed to do the protocol. RFC 6334 [28] defines an AFTR (Address Family Transition Router) name DHCPv6 option that maps to an EndpointName parameter in the Device:2 data model[7].

EndpointName is a variable length field, containing a Fully Qualified Domain Name that refers to the AFTR the client is requested to establish a connection with. EndpointName can be assigned statically (e.g. present in the factory default configuration or set by the Controller) or dynamically (via DHCPv6). If both statically and dynamically assigned, then the EndpointAssignmentPrecedence parameter indicates whether it is the static configuration or the DHCPv6 configuration that is actually applied to EndpointName.

EndpointAddress is a 128 bit field, containing one IPv6 address. The tunnel EndpointAddress specifies the location of the remote tunnel endpoint, expected to be located at an AFTR. EndpointAddress can be assigned statically (e.g. present in the factory default configuration or set by the Controller) or dynamically (via DNS lookup when EndpointName is set). If both statically and dynamically assigned, then the EndpointAssignmentPrecedence parameter indicates whether it is the static configuration or the DHCPv6-derived configuration that is actually applied to EndpointAddress.

When EndpointName is assigned, the name is looked up (resolved) and the corresponding IPv6 address is set in EndpointAddress.

When DS-Lite is running in the CPE, the NAT function is disabled between the LAN and DSLite interface.

## VII.1 Internal Treatment of IPv4 Packets

Since a device can have multiple upstream and multiple downstream interfaces, the model supports a logical representation of the internal virtual DS-Lite IPv4 interface according to the general pattern described in Annex B.

The internal virtual DS-Lite IPv4 interface is modeled as (*TunnelInterface,TunneledInterface*).

The IPv4Forwarding entries will route traffic between the downstream IPv4 interfaces and the DS-Lite IPv4 interface. IPv6Forwarding entries are unaffected.

[7] *Introduced in Amendment 2*

Figure 39 shows the flow of tunneled DS-Lite traffic through the downstream, upstream, and logical tunnel interfaces. Noted in the figure are sample values for the various *IP.Interface* entries that would be needed.

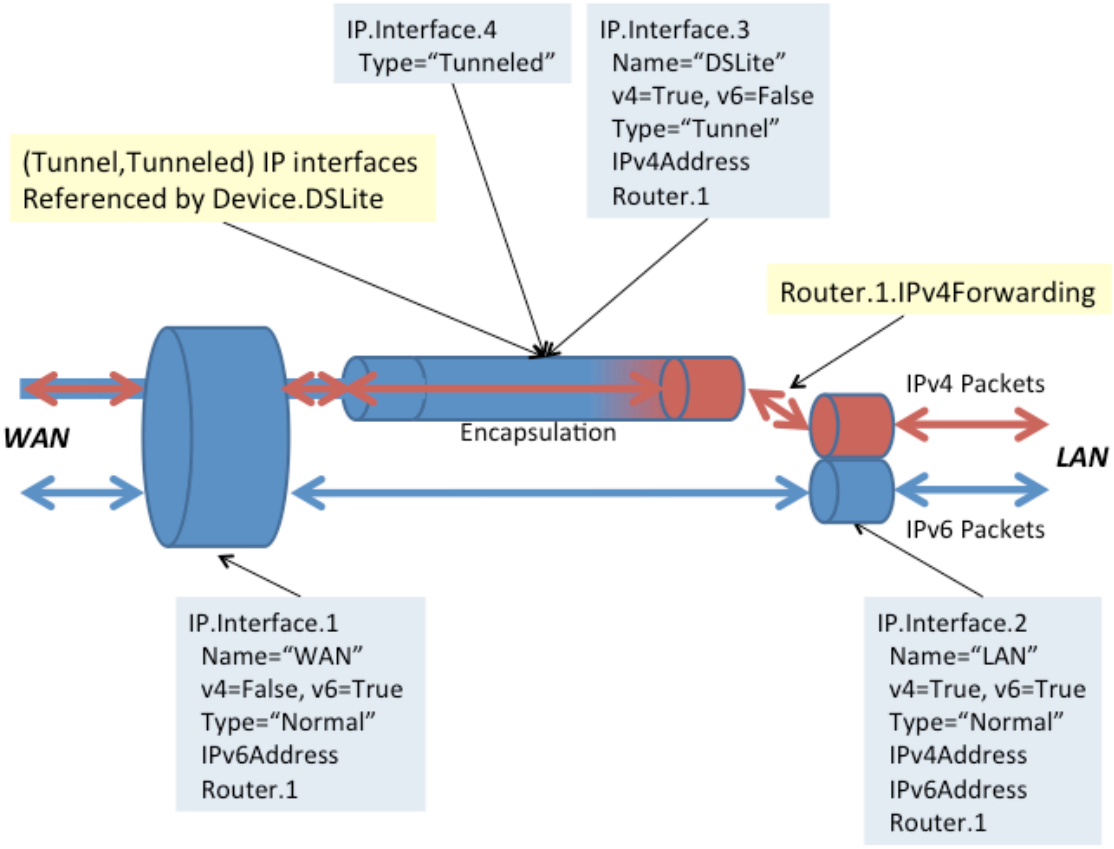



**Figure 39 – Sample DS-Lite Routing and Forwarding**

# Appendix VIII Advanced Firewall Example Configuration

This Appendix presents an advanced firewall example that illustrates settings corresponding to the following predefined Firewall.Config levels:

- **High**: The firewall implements the “Traffic Denied Inbound” and “Minimally Permit Common Services Outbound” components of the ICSA residential certification's Required Services Security Policy [34]. If DoS and vulnerability protections are implemented [33], these are enabled.
- **Low**: All Outbound traffic and pinhole-defined Inbound traffic is allowed. If DoS and vulnerability protections are implemented [33], these are enabled.

```
Firewall.
      Enable = true
      Config = "Advanced"
      AdvancedLevel = Firewall.Level.1
      Type = "Stateful"

Firewall.Level.1.
      Name = "High"
      Description = "Deny Inbound and minimally permit Outbound"
      Order = 1
      Chain = Firewall.Chain.1
      DefaultPolicy = "Drop"

Firewall.Level.2.
      Name = "Low"
      Description = "Allow all Outbound and pinhole-defined Inbound"
      Order = 2
      Chain = Firewall.Chain.2
      DefaultPolicy = "Drop"

Firewall.Chain.1.
      Name = "High (Deny Inbound and minimally permit Outbound)"
      Creator = "Defaults"
      Rule.1.
            Order = 1
            Description = "Telnet"
            Target = "Accept"
            DestInterface = IP.Interface.1   # upstream facing IP interface
            Protocol = 6                     # TCP
            DestPort = 23
      Rule.2.
            Order = 2
            Description = "FTP"
            Target = "Accept"
            DestInterface = IP.Interface.1   # upstream facing IP interface
            Protocol = 6                     # TCP
            DestPort = 21
      Rule.3.
            Order = 3
            Description = "HTTP"
            Target = "Accept"
```

```
            DestInterface = IP.Interface.1   # upstream facing IP interface
            Protocol = 6                     # TCP
            DestPort = 80
      Rule.4.
            Order = 4
            Description = "HTTPS"
            Target = "Accept"
            DestInterface = IP.Interface.1   # upstream facing IP interface
            Protocol = 6                     # TCP
            DestPort = 443
      Rule.5.
            Order = 5
            Description = "SMTP"
            Target = "Accept"
            DestInterface = IP.Interface.1   # upstream facing IP interface
            Protocol = 6                     # TCP
            DestPort = 25
      Rule.6.
            Order = 6
            Description = "DNS"
            Target = "Accept"
            DestInterface = IP.Interface.1   # upstream facing IP interface
            Protocol = 17                    # UDP
            DestPort = 53
      Rule.7.
            Order = 7
            Description = "POP3"
            Target = "Accept"
            DestInterface = IP.Interface.1   # upstream facing IP interface
            Protocol = 6                     # TCP
            DestPort = 110
      Rule.8.
            Order = 8
            Description = "IMAP"
            Target = "Accept"
            DestInterface = IP.Interface.1   # upstream facing IP interface
            Protocol = 6                     # TCP
            DestPort = 143



Firewall.Chain.2.
      Name = "Low (Allow all Outbound and pinhole-defined Inbound)"
      Creator = "Defaults"
      Rule.1.
            Order = 1
            Description = "Outbound"
            Target = "Accept"
            DestInterface = IP.Interface.1   # upstream facing IP interface
      Rule.2.
            Order = 2
            Description = "Allow IPsec AH"
            Target = "Accept"
            SourceInterface = IP.Interface.1 # upstream facing IP interface
            IPVersion = 6                    # IPv6
            Protocol = 51                    # AH
      Rule.3.
```

```
            Order = 3
            Description = "Allow IPsec ESP"
            Target = "Accept"
            SourceInterface = IP.Interface.1 # upstream facing IP interface
            IPVersion = 6                    # IPv6
            Protocol = 50                    # ESP
      Rule.4.
            Order = 4
            Description = "Allow IPsec key exchange"
            Target = "Accept"
            SourceInterface = IP.Interface.1 # upstream facing IP interface
            IPVersion = 6                    # IPv6
            Protocol = 17                    # UDP
            DestPort = 500
      Rule.5.
            Order = 5
            Description = "UPnP Port Mapping"
            Target = "TargetChain"
            TargetChain = Firewall.Chain.3
            SourceInterface = IP.Interface.1 # upstream facing IP interface
      Rule.6.
            Order = 6
            Description = "UPnP IPv6 Firewall"
            Target = "TargetChain"
            TargetChain = Firewall.Chain.4
            SourceInterface = IP.Interface.1 # upstream facing IP interface
      Rule.7.
            Order = 7
            Description = "User Interface"
            Target = "TargetChain"
            TargetChain = Firewall.Chain.5
            SourceInterface = IP.Interface.1 # upstream facing IP interface

Firewall.Chain.3.
      Name = "UPnP Port Mapping (dynamic rules)"
      Creator = "PortMapping"
      Rule.1.
            Order = 1
            Description = "SSH"
            Target = "Accept"
            SourceInterface = IP.Interface.1 # upstream facing IP interface
            IPVersion = 4                    # IPv4
            Protocol = 6                     # TCP
            DestPort = 22

Firewall.Chain.4.
      Name = "UPnP IPv6 Firewall (dynamic rules)"
      Creator = "WANIPv6FirewallControl"
      Rule.1.
            Order = 1
            Description = "HTTP"
            Target = "Accept"
            SourceInterface = IP.Interface.1 # upstream facing IP interface
            IPVersion = 6                    # IPv6
            Protocol = 6                     # TCP
            DestIP = 1080:0:0:800::1
            DestPort = 80
```

```
Firewall.Chain.5.
      Name = "User Interface"
      Creator = "UserInterface"
      Rule.1.
            Order = 1
            Description = "SMTP server"
            Target = "Accept"
            SourceInterface = IP.Interface.1 # upstream facing IP interface
            IPVersion = 4                    # IPv4
            Protocol = 6                     # TCP
            DestIP = 192.168.1.4
            DestPort = 25
      Rule.2.
            Order = 2
            Description = "DMZ"
            Target = "Accept"
            SourceInterface = IP.Interface.1 # upstream facing IP interface
            IPVersion = 4                    # IPv4
            DestIP = "192.168.1.5" # IPv4 address of LAN device that recvs
                                   # all unsolicited inbound IPv4 traffic
```

# Appendix IX IPsec Theory of Operation

See Annex B for general information on how tunneling is modeled.

The Device:2 data model includes an IPsec (RFC 4301 [35]) object that supports the configuration of Encapsulating Security Payload (ESP; RFC 4303 [37]) and Authentication Header (AH; RFC 4302 [36]) in tunnel mode (Section 3.2/RFC 4301). Use of IKEv2 (RFC 5996 [38]) is assumed. The IPsec object does not currently support static configuration of tunnels and child Security Associations (SAs).

Figure 40 illustrates the main IPsec objects and their relationships.

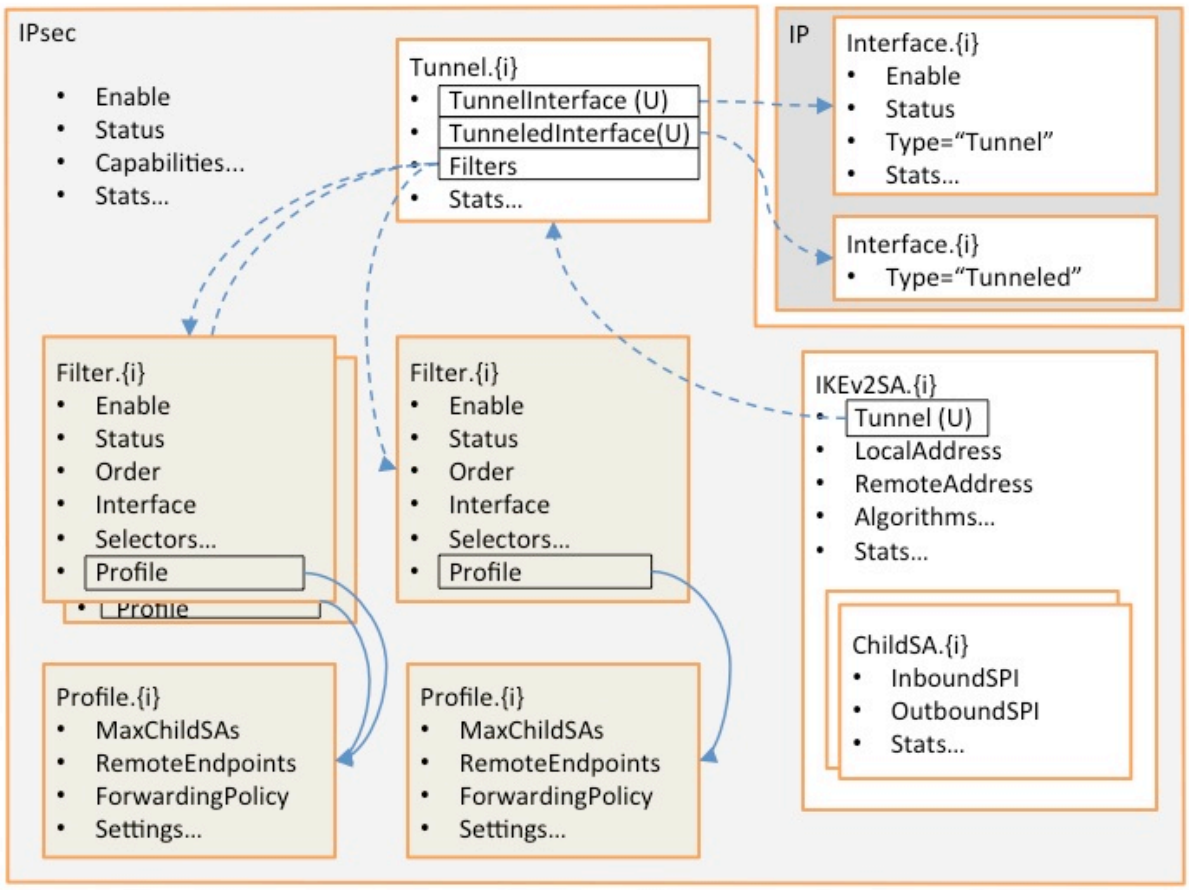



**Figure 40 – IPsec Data Model Objects**

In the Figure, instances of the colored objects (*Filter.{i}* and *Profile.{i}*) are created and populated by the Controller. Instances of all other objects are handled by the CPE as IPsec tunnels are created and deleted. References between objects are shown:

- Solid lines indicate references that are populated by the Controller, and dashed lines indicate references that are handled by the CPE.
- A reference marked "(U)" is a unique key, which implies a 1-1 relationship, e.g. only one *Tunnel* instance can reference a given (*Tunnel,Tunneled*) *IP.Interface* pair.

- Other references imply n-1 relationships, e.g. multiple *Filter* instances can reference a given *Profile* instance.

Typical usage is as follows:

- The factory default configuration can contain static instances of the various objects.
- The Controller creates and configures *Filter* and *Profile* instances. *Filter* instances model IPsec Security Policy Database (SPD) selection criteria and *Profile* instances model SPD processing info. Each *Filter* instance references a *Profile* instance so a single *Profile* instance can, if desired, be shared by several *Filter* instances.
- When the Controller enables a *Filter* instance, the CPE determines whether a new tunnel is needed in order to carry the traffic that matches that filter. It is possible that an existing tunnel can carry the traffic.
- If a new tunnel is needed, the CPE immediately creates a *Tunnel* instance that references a newly-created (*Tunnel,Tunneled*) *IP Interface* pair. This corresponds exactly to the general tunneling approach that is described in Annex B.
- Each *Tunnel* instance also references all of the currently-enabled *Filter* instances that require it to exist.
- Classification and forwarding rules can now be defined, regardless of whether the tunnels have yet been established. *ForwardingPolicy* is both a QoS *Classification* result and an IPsec *Filter* result (it's in the *Policy* table), and so can, as explained in Annex B, affect the forwarding decision and thus whether or not a given packet will be en-tunneled or de-tunneled.
- When a tunnel needs to become active, e.g. as a result of traffic that matches one of the *Filter* instances, the CPE will establish it and will create the appropriate *IKEv2SA* and *ChildSA* objects.
- When a tunnel no longer needs to be active, the CPE will delete the *ChildSA* and *IKEv2SA* objects. This will affect the status of the *Tunnel* instance and (*Tunnel,Tunneled*) *IP Interface* pair but will not delete them.

The remainder of this Appendix consists of a brief summary of the various IPsec data model objects.

## IX.1 IPsec

The top-level object has an *Enable* parameter that enables and disables the IPsec sub-system, various capability parameters, e.g. supported encryption algorithms, and global IPsec statistics.

## IX.2 IPsec.Filter

The *Filter* table models IPsec Security Policy Database (SPD) selection criteria. Refer to Section 4.4.1/RFC 4301 [35] for further details.

SPD filtering is performed for all packets that might need to cross the IPsec boundary. Refer to Section 3.1/RFC4301 for further details. Given that IPsec operates at the IP level, this means that SPD filtering conceptually occurs after bridging and before routing.

This table is conceptually quite similar to the QoS Classification table in that entries are ordered, associated with an ingress interface, include selection criteria, and specify the action to be taken for matching packets.

Instances of the *Filter* table can be created statically by the CPE, or can be created and deleted by the Controller as needed. Each instance includes the following (this is not a complete list):

- *Enable*: to enable and disable the entry.
- *Status*: to indicate the status of the entry.
- *Order*: to control and indicate the order of the entry.
- *Interface*, *AllInterfaces*: to control and indicate with which interfaces the entry is associated.
- *DestIP*: to select packets by destination IP address.
- *SourceIP*: to select packets by source IP address.
- *Protocol*: to select packets by IP protocol.
- *DestPort*: to select packets by destination port.
- *SourcePort*: to select packets by source port.
- *Discard*: whether to discard matching packets.
- *Profile*: the Profile instance that governs how non-discarded matching packets will be treated.

## IX.3 IPsec.Profile

The *Profile* table models IPsec Security Policy Database (SPD) processing info. Refer to Section 4.4.1/RFC 4301 [35] for further details. Each *Filter* instance references a *Profile* instance. It would be possible to include the processing info directly in each *Filter* instance, but use of a separate table allows *Profile* entries to be shared between *Filter* instances.

Instances of the *Profile* table can be created statically by the CPE, or can be created and deleted by the Controller as needed. Each instance includes the following (this is not a complete list):

- *MaxChildSAs*: the maximum number of Child SAs per IKEv2 session (and therefore per IPsec tunnel); this provides a simple way of controlling the extent to which existing tunnels can be re-used.
- *RemoteEndpoints*: an ordered list of remote tunnel endpoints that are to be used when establishing an IPsec tunnel corresponding to this *Profile* instance.
- *ForwardingPolicy*: an opaque (Controller-chosen) value that provides a feed-forward mechanism that allows the SPD filtering decision to affect the forwarding decision. QoS classification uses the same mechanism.
- *Protocol*: the “child” security protocol, i.e. AH or ESP.
- *IKEv2AuthenticationMethod*: a reference to a CPE certificate or other CPE credentials.

- *IKEv2AllowedEncryptionAlgorithms* (etc): encryption algorithm that IKEv2 is permitted to negotiate; also several other "allowed" parameters that define acceptable IKEv2, AH and ESP algorithms.
- *DSCPMarkPolicy* (etc): various settings that govern how packets should be tunneled.

## IX.4 IPsec.Tunnel

The *Tunnel* table that models IPsec tunnels. Instances are created and deleted by the CPE as needed. A (*Tunnel,Tunneled*) *IP Interface* pair[8] is always created at the same time as an IPsec *Tunnel* instance and has the same lifetime; the *Tunnel IP Interface* contains generic IP interface settings, e.g. *Enable*, *Status* and generic *Stats*, and the IPSec *Tunnel* instance contains IPsec-specific settings, e.g. additional *Stats*.

## IX.5 IPsec.IKEv2SA

Each entry in the *IKEv2SA* table models a single IKEv2 SA pair and uniquely references a *Tunnel* instance. Unlike *Tunnel* instances, which exist regardless of whether the tunnel is active, *IKEv2SA* instances exist only when the IKEv2 SA pair exists, i.e. they exist only when the tunnel is active.

## IX.6 IPsec.IKEv2SA.ChildSA

The *ChildSA* table models child SA pairs. It is a child of the corresponding *IKEv2SA* instance and so exists only when the *IKEv2SA* instance exists.

[8] *i.e. an IP Interface instance with Type = "Tunnel", and another IP Interface instance with Type = "Tunneled".*

# Appendix X ETSI M2M Remote Entity Management Theory of Operation

*NOTE* - ETSI currently only endorses TR-069 [2] for management of M2M devices, but the principles would also apply for USP, even if the protocol is not mentioned in this appendix.

Figure 41 below depicts the high level ETSI M2M functional architecture defined in section 4 of ETSI TS 102 690 [39]. The Data Models defined [41] are used within CWMP enabled Devices and Gateways within the Device and Gateway domain.

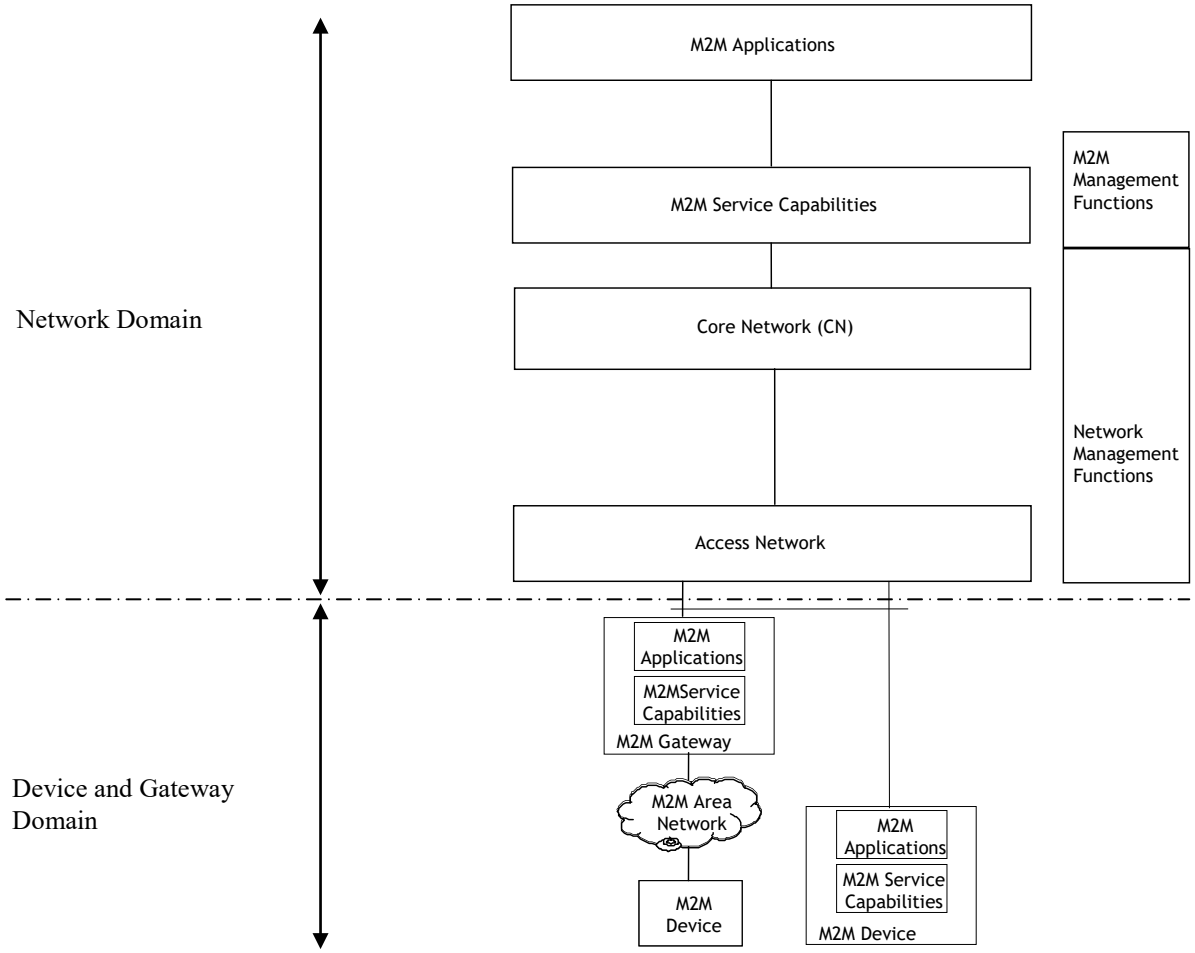



**Figure 41 – ETSI High Level Functional Architecture**

Within the Device and Gateway Domain, the M2M Device and Gateway contains 2 functional components as defined in the ETSI M2M Functional Architecture [39]:

- M2M Service Capabilities: M2M functions that are to be shared by different M2M Applications.
- M2M Applications: Applications that run the service logic and use M2M Service Capabilities.

Interactions between components within the ETSI architecture are defined using reference points. Figure 42 below illustrates the Service Capability Layer (SCL) mId reference point that is of interest. A full explanation of the SCL reference points is provided in section 5 of the ETSI M2M Functional Architecture [39].

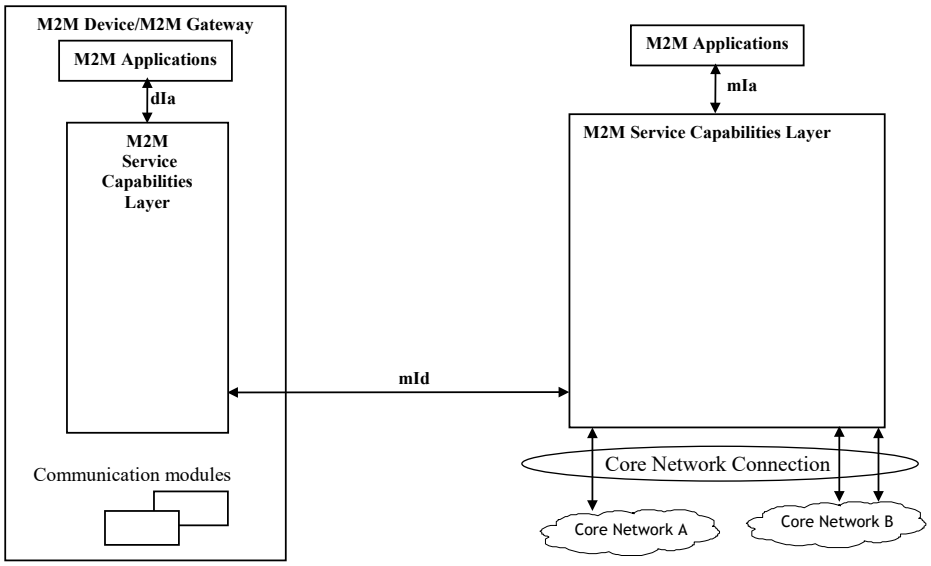



**Figure 42 – M2M SCL Functional Architecture Framework**

The M2M Device or Gateway SCL provides capabilities (functionality) for the following areas:

- Application Enablement (xAE)
- Generic Communication (xGC)
- Reachability, Addressing and Repository (xRAR)
- Communication Selection (xCS)
- Remote Entity Management (xREM)
- SECurity (xSEC)
- History and Data Retention (xHDR)
- Transaction Management (xTM)
- Compensation Broker (xCB)
- Telco Operator Exposure (xTOE)
- Interworking Proxy (xIP)

*NOTE - The « x » designates a capability is used in the context of the Device (D) or Gateway (G).*

The Data Model in [40] reflects the device management objects and parameters necessary to implement xREM functionality across the mId reference point as defined in Annex E of the ETSI Functional Architecture [39] is depicted in Figure 43. In this instance, the Device Mgmt Client is considered a CWMP endpoint interface and the Device Mgmt Server is considered the ACS interface. In most situations, these endpoints and servers have an interface between the native Device, Gateway or Server environment and the SCL. In addition, the dIa reference point, using RESTful procedures, is used to discover M2M D' Devices and M2M Applications as well as proxy selected xREM management functions.

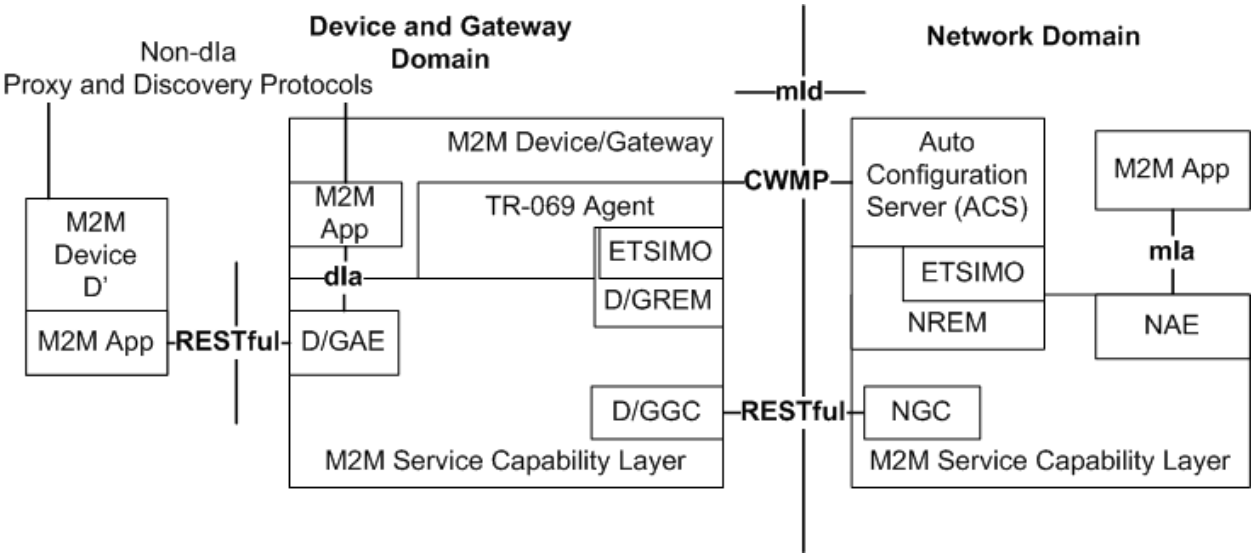



**Figure 43 – M2M REM Service Capability**

*NOTE* - The mId reference point in this scenario would support CWMP for the exchange of "mgmtObjs" using the xREM procedures between SCLs while continuing to support the ETSI RESTful procedures (e.g., container management) for the exchange of other resources across the mId reference point.

Within the ESTI M2M Functional Architecture, the xREM is responsible for the following management functions:

- General Management: Provides retrieval of information related to the M2M Device or Gateway that hosts the ETSI M2M Service Capability Layer (SCL).
- Configuration Management: Provides configuration of the M2M Device or Gateway's capabilities in order to support ETSI M2M Services and Applications.
- Diagnostics and Monitoring Management: Provides diagnostic tests and retrieves/receives alerts associated with the M2M Device or Gateway that hosts the SCL.
- Software Management: Maintains software associated with the SCL and M2M services.
- Firmware Management: Maintain firmware associated with the M2M Device or Gateway that hosts the SCL.
- Area Network Management: Maintains devices on the M2M Area Network associated with the SCL.
- SCL Administration: Provides administration capabilities in order to configure and maintain a SCL within the M2M Device or Gateway.

Within the customer premises, equipment is categorized within the ETSI M2M framework as a:

- M2M Gateway: A Gateway that runs M2M Application(s) using M2M Service Capabilities.
- M2M Device: A Device that runs applications using M2M capabilities and network domain functions. Depending on M2M capabilities of the M2M Device, the M2M Device is defined as a:
  - Device (D): provides M2M Service Capabilities (DSCL) that communicates to an NSCL using the mId reference point and to DA using the dIa reference point
  - Device' (D'): hosts a Device Application (DA) that communicates to a GSCL using the dIa reference point. D' does not implement ETSI M2M Service Capabilities
- Non-ETSI M2M complaint device (d): A device that connects to a SCL through the SCL's Interworking Proxy capability.

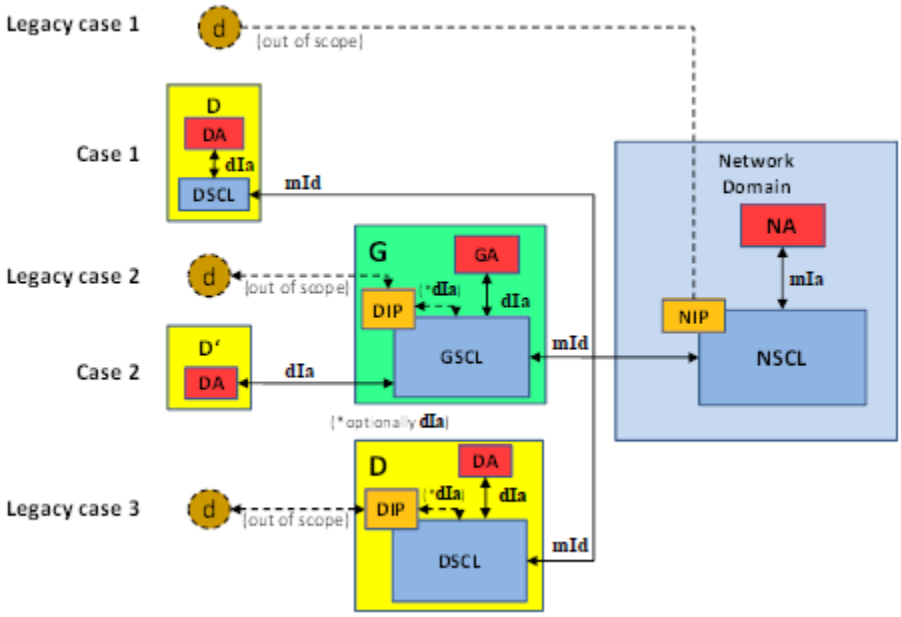



**Figure 44 - ETSI M2M Devices and Gateways**

## X.1 ETSI M2M Area Networks

In the ETSI framework D' and d Devices that connect to a SCL within a M2M Device or Gateway are said to be "attached devices" and are organized by M2M Area Networks within the SCL. The mechanism that a M2M Gateway uses to identify M2M Area Networks and their associated devices is implementation specific.

## X.2 Device:2 Data Model and Functionality for ETSI M2M REM

Annex B of the ETSI M2M Functional Architecture [39] provides a cross reference between the xREM management functions and the object instances and RPCs required to implement the management functionality. The following is a summary of the objects, services, components, RPCs and optional TR-069 functionality required by the ETSI M2M xREM solution.

The ETSI M2M xREM solution in Annex E of the ETSI M2M Managed Objects [40] defines a cross reference of the following ETSI resources to the existing Device:2 Data Model objects. These ETSI resources are:

- etsiDeviceInfo
- etsiDeviceCapability
- etsiMemory
- etsiTrapEvent
- etsiPerformanceLog
- etsiFirmware
- etsiSoftware
- etsiReboot

The implementation of these resources the use of the following objects from the data model:

- DeviceInfo.
- WiFi.
- SmartCardReaders.
- USB.
- HomePlug.
- MoCa.
- UPA.
- UPnP.
- Hosts.
- SoftwareModules.
- FaultMgmt. (Use for etsiTrapEvent)
- SelfTestDiagnostics.
- DeviceInfo.VendorLogFile. (Use for etsiPerformanceLog)
- ManagementServer.EmbeddedDevice.
- ManagementServer.VirtualDevice.

### X.2.1 TR-069 Functionality for ETSI M2M REM

In addition to the mandatory RPCs defined in TR-069 [2], the ETSI M2M xREM solution requires that a M2M Device or Gateway implement the following optional RPCs according to Section 9.2.1.11 of [39]:

- Upload method
- ScheduleDownload method
- ScheduleInform method
- ChangeDUState method
- FactoryReset method

## X.3 Device:2 Data Model and Functionality for ETSI M2M REM

In addition to reusing objects and parameters, the ETSI M2M xREM solution defines extensions to the resource model for the following ETSI resources by defining extensions to the data model for the following ETSI resources:

- etsiSclMo
- etsiAreaNwkInfo
- etsiAreaNwkDeviceInfo

These resources provide administration of the SCL in order for the SCL in the Device or Gateway to communicate with SCLs in the network. In addition, these resources provide administration of the SCL for M2M Devices within the local M2M area network attached to a Device or Gateway in order to communicate with associated network SCLs.

The ETSI M2M Services Device model defines the ETSIM2M service in support of the xREM functionality.

### X.3.1 M2M Service SCL Execution Environment

CPEs that provide software execution capabilities have the option to implement the Gateway Service Capabilities Layer and Gateway Applications as software modules. When a SCL is implemented as a software module, each instance of the GSCL and GA would be represented as individual Deployment Units with the associated software and configuration files. For the GSCL the vendor configuration file could contain configuration elements (e.g., M2M Node Id, NSCL List) that would be returned from or necessary to perform the M2M Service Bootstrap and Service Connection Procedures.

### X.3.2 ETSIM2M Object

The ETSIM2M objects provide administration of the SCL instantiated within a Device or Gateway.

The primary administration functions of the service are to:

- Maintain the set of Network SCLs (NSCL) that the M2M Device or Gateway SCL is registered.
- Maintain the set of NSCLs to which the M2M Device or Gateway will "announce" local resources.
- Maintain a list of Store and Forward (SAF) policies associated with the access network provider for message handling between M2M Devices in the area network and the NSCL.
- Maintain a list of Store and Forward (SAF) policies associated with the access network provider for message handling between the gateway and the NSCL.
- Maintain a list of Store and Forward (SAF) policies associated with the M2M service provider for message handling between M2M Devices in the area network and the NSCL.
- Maintain a list of Store and Forward (SAF) policies associated with the M2M service provider for message handling between the gateway and the NSCL.
- Discovery and Maintenance of M2M Area Networks.
- Discovery and Maintenance of M2M Devices.

*NOTE - As a SCL instance within a M2M Device or Gateway is associated with one M2M service provider, the M2M Device or Gateway is capable of maintaining multiple SCL instances.*

#### X.3.2.1 M2M Service Bootstrap and Service Connection Procedures

In the ETSI M2M system, the M2M (Device or Gateway) Node must establish the capability to connect with a M2M Network Node before the SCLs are permitted to be registered using M2M Service Bootstrap and Service Connection procedures.

The M2M Service Bootstrap and Service Connection procedures are defined in section 8.2 of the ETSI M2M Functional Architecture [39] and describe how some of the credentials are shared and obtained in order to establish a connections (e.g, HTTP TLS-PSK) during the exchange of RESTFul information over the mId reference point.

#### X.3.2.2 Rules for Instantiating a SCL Instance

A M2M Node is not modeled as a device management entity but is considered a logical representation of the M2M components in the M2M Device, M2M Gateway or the M2M Core. Such components include:

- One instance of a SCL
- An optional M2M Service Bootstrap procedure

- A M2M Service Connection procedure

A M2M Node is identified by a globally unique identifier, the M2M-Node-ID.

In addition to the logical representation of a M2M Node, the following are constraints of a M2M Node that reflect on why a M2M Device or Gateway would instantiate multiple SCL instances:

- A M2M Node is owned by one M2M Service Provider.
- A M2M Node is instantiated upon M2M Bootstrap procedure or pre-provisioning the M2M Device or Gateway with a M2M Service Provider.
- Multiple M2M Nodes MAY be instantiated on the same M2M Device or Gateway by performing multiple M2M Bootstrap procedures either with the same M2M Service Provider or with different M2M Service Providers.

#### X.3.2.3 SCL Addressing

When a SCL is instantiated the SCL is provided a SCL-ID using the M2M Service Bootstrap procedure or through an out-of-band mechanism. Table 7.1 of the ETSI M2M Functional Architecture [39] describes the characteristics of the SCL-ID.

When a M2M Device or Gateway SCL registers with a NSCL, the NSCL maintains the following information in its resource tree for the SCL that allows the NSCL to identify and contact the M2M Device or Gateway SCL:

- SCL-ID that globally unique and MAY be the same as the M2M-Node-ID.
- M2MPoCs contactInfo of the M2M Device or Gateway SCL – This MAY be the FQDN, IP Address and port information or it MAY be other information that the M2M Service Provider can use to ask the network access provider for an IP Address.

#### X.3.2.4 SCL Registration

In order to communicate requests between the M2M Device or Gateway SCL and the NSCL, the M2M Device or Gateway SCL registers with the NSCL. Section 9.3.2.6.2 of the ETSI M2M Functional Architecture [39] describes the registration process including how attributes such as the SCLID, search strings and expiration times are provisioned. In order for a M2M Device or Gateway SCL to register with the NSCL, the M2M Device or Gateway SCL must be provisioned with a list of potential NSCLs that the M2M Device or Gateway SCL is registered. In addition to the list of NSCLs, the M2M Device or Gateway SCL also has parameters to manage when a M2M Device or Gateway SCL re-registers with the NSCL. The M2M Device or Gateway SCL also has the capability to be requested to re-register with the NSCL through its TR-069 interface.

#### X.3.2.5 Discovery of M2M Devices through the SCL

Using the control plane, the M2M Device or Gateway SCL provides the capability to return a list of resources that the M2M Device or Gateway has discovered. Filtering MUST be performed on a subset of the offered resources' attributes using a query string. A match, that MAY include ranges, is performed on the query string, and a successful response is returned with a URI(s) list for resources that contains the matching attributes. Section 9.3.2.27 of the ETSI M2M Functional

Architecture [39] describes this procedure. The M2M Device or Gateway MAY be provisioned through the TR-069 interface to either limit the number of URIs discovered by the device or define the maximum size allowed for a discovery result.

#### X.3.2.6 De/Announcing M2M Devices through the SCL

One capability of the M2M Device or Gateway SCL control plane is to announce or de-announce M2M resources (e.g., access rights, applications) to NSCL(s) to which the M2M Device or Gateway SCL has registered if the SCL is contained within the "AnnounceToSCLList". Section 9.3.2.28 of the ETSI M2M Functional Architecture [39] describes this procedure. The "AnnouncedToSCLList" is maintained through the TR-069 interface.

#### X.3.2.7 SCL Store and Forward Policies

The M2M Device or Gateway SCL is responsible for handling requests from an attached M2M Device or itself and the NSCL. The handling of the requests is based on criteria within the request (e.g., Request category [RCAT], Tolerable Request Processing Delay [TRPDT]) as well as conditions within the M2M Device or Gateway SCL (e.g., pending requests, access network availability).
There are two types of SCL store and forward (SAF) policies:

- Access Network Provider SAF Policies
- Service Provider SAF Policies

The SAF policies are organized into instances of Policy sets. The selection of which Policy sets are used by the M2M Device or Gateway SCL is determined by the PolicyScope attribute of the Policy set.

Section 9.3.1.5 of the ETSI M2M Functional Architecture [39] describes this procedure. These policies are maintained through the TR-069 interface.

##### X.3.2.7.1 Access Network Provider SAF Policies

Access Network Provider SAF policies are used by M2M Device or Gateway SCLs to determine if an Access Network is to be used when forwarding requests from the M2M Device or Gateway SCL to the NSCL. The determination of which Access network to use is based on:

- Schedule of RCAT values versus time: The M2M Device or Gateway SCL is provisioned with information from the NSCL for the access network provider regarding when it is appropriate to forward requests of a given RCAT value.
- Blocking of access attempts after failure to establish connectivity: The M2M Device or Gateway SCL is provisioned with information from the NSCL for the access network provider regarding the period of time over which attempts to establish connectivity over its access network are not appropriate after the previous attempt to establish connectivity over the corresponding access network has failed. The period of time to block attempts to establish connectivity can be a function of the number of consecutive previous attempts to establish connectivity over this access network.

*NOTE* - An Access Network Provider SAF is identified from the Access Network Provider name parameter.

#### X.3.2.7.2 M2M Service Provider SAF Policies

M2M Service Provider Store and Forward (SAF) policies are used by M2M Device or Gateway SCLs to determine to forward a request to NSCL. The determination if the request is forwarded is based on the:

- Wait time as function of number of pending requests: The M2M Device or Gateway SCL is provisioned with information from the NSCL for the service provider regarding how many pending requests of a given range of RCAT values are sufficient to forward the aggregated request to the NSCL. The ranges of RCAT values for different policies cannot overlap.
- Wait time as function of amount of pending request data: The M2M Device or Gateway SCL is provisioned with information from the NSCL for the service provider regarding a threshold of consumed storage (memory) in the M2M Device or Gateway SCL that is needed to buffer data for pending requests of a given range of RCAT values. The ranges of RCAT values for different policies cannot overlap.
- Selection among appropriate access networks: The M2M Device or Gateway SCL is provisioned with information from the NSCL for the service provider regarding how to select an access network for making an attempt to establish connectivity from an ordered list of possible access networks for a given range of RCAT values. The ranges of RCAT values for different policies cannot overlap.
- Default values for TRPDT and RCAT: The M2M Device or Gateway SCL is provisioned with information from the NSCL for the service provider regarding the TRPDT and RCAT values to use if they are not provided by the request issuer.

### X.3.2.8 Area Network Discovery and Maintenance

The M2M Device or Gateway SCL discovers properties of instances of M2M Area Networks as well as the Devices (D', d) associated with a M2M Area Network. A M2M Area Network is a logical entity in that an instance of an Area Network can span one or more physical interfaces of the M2M Device or Gateway. In addition, a M2M Gateway can provide connectivity to more than one instance of the same type of M2M Area Network. Examples of M2M Area Networks include: Personal Area Network technologies such as IEEE 802.15.x, Zigbee, Bluetooth, IETF ROLL, ISA100.11a or local networks such as PLC, M-BUS, Wireless M-BUS and KNX.

A M2M Area Network is maintained as instances of an AreaNwkInstance. Each AreaNwkInstance maintains opaque properties of the Area Network using Property instances of name/value pairs. In addition, the AreaNwkInstance also maintains a list of references to instances of AreaNwkDeviceInfoInstance table that are associated with the Area Network.

### X.3.2.9 M2M Device Discovery and Maintenance

The M2M Device or Gateway maintains a list of discovered M2M Devices (D', d) that are attached to the SCL. A discovered M2M Device that is associated with more than one AreaNwkInstance is represented as multiple instances of AreaNwkDeviceInfoInstance objects.

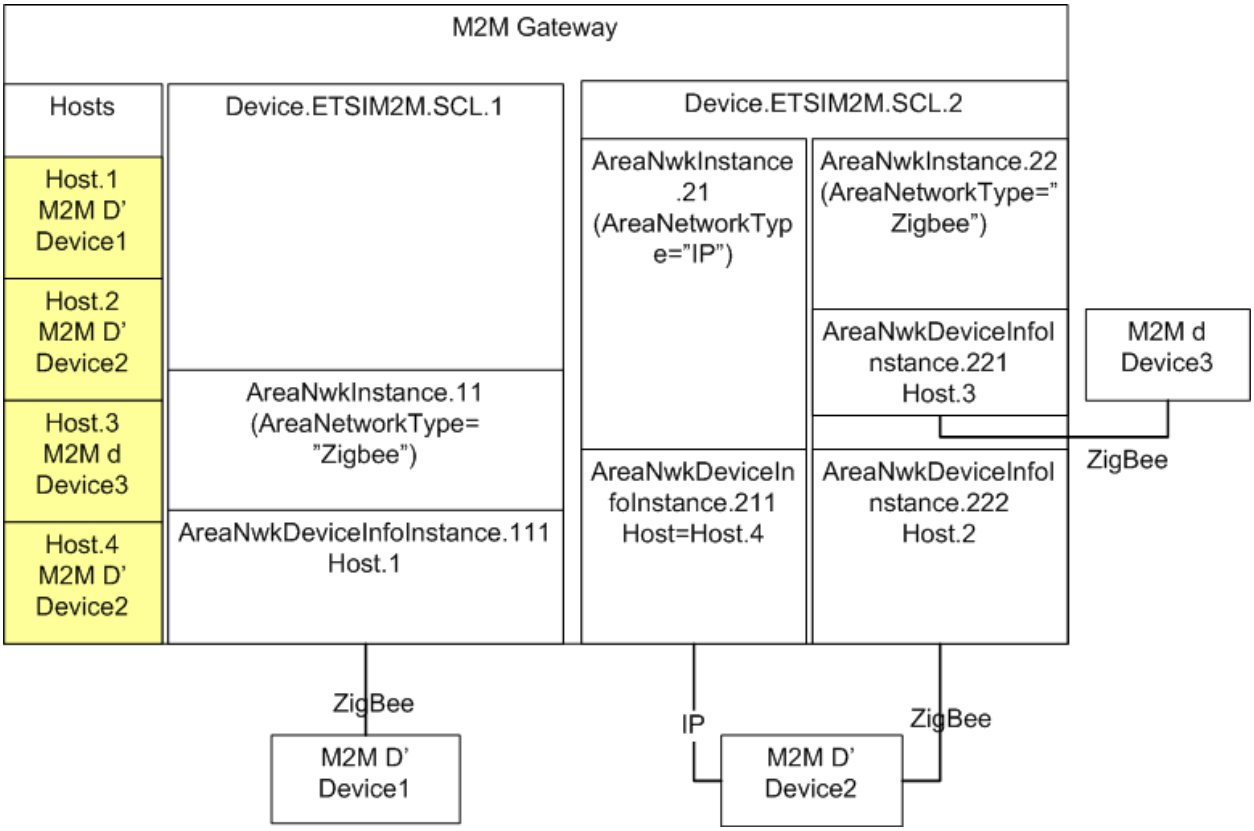



**Figure 45 - Example M2M Network**

In Figure 45, an M2M Gateway has two (2) SCL instances that manage three (3) M2M Devices. Each M2M Device is represented in the Root Data Model's Hosts.Host table. The M2M Devices are represented by the AreaNwkDeviceInfoInstance object that was discovered within a context of an AreaNwkInstance of a SCL. As a M2M Device is capable of being discovered through multiple M2M Area Networks, different instances of the AreaNwkDeviceInfoInstance could reference the same or different Host table entry.

Each AreaNwkDeviceInfoInstance maintains a reference to an AreaNwkInstance object as well as properties specific to the device and area network association (e.g., SleepInterval). In addition, each AreaNwkDeviceInfoInstance maintains opaque properties of the device using Property instances of name/value pairs.

### X.3.2.9.1 M2M Device Discovery and Maintenance

M2M Devices are able to be managed through the TR-069 Embedded Object and Virtual Device Proxy management capabilities. In these scenarios the AreaNwkDeviceInfoInstances are known as Discovered Devices.

In the scenario where a M2M Device (D', d) is discovered as part of an Embedded or Virtual Device, the AreaNwkDeviceInfoInstance is maintained as an item in the DiscoveryProtocolReference parameter of the Embedded or Virtual Device using one or more of the protocols listed in the DiscoveryProtocol parameter. Figure 46 describes the scenario where the M2M Devices are discovered using the ETSI-M2M protocols.

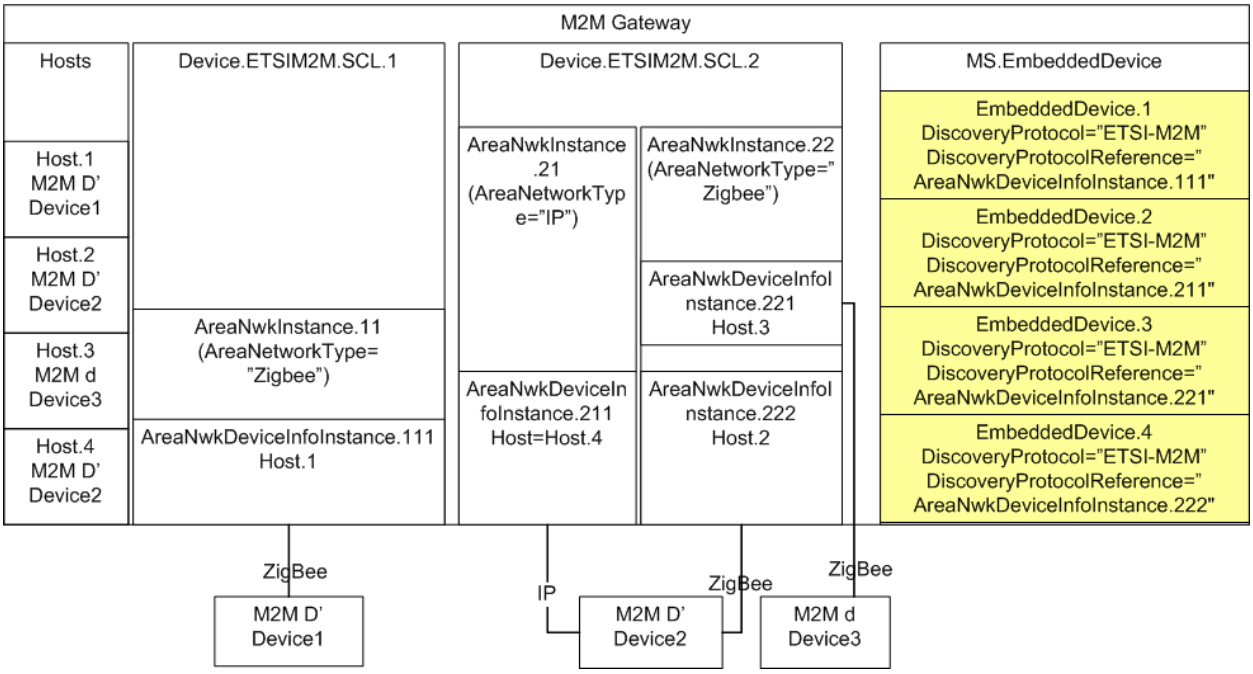



**Figure 46 - M2M Device Discovery for Proxy Management**

### X.3.2.10 SCL Configuration

The ETSI M2M Data Model includes the capability to provision the SCL with objects and parameters necessary for the SCL to host resources and transfer messages between M2M Devices and Gateway Applications and the NSCL. This section describes the minimal configuration necessary for an SCL to:

Host resources

Transfer messages

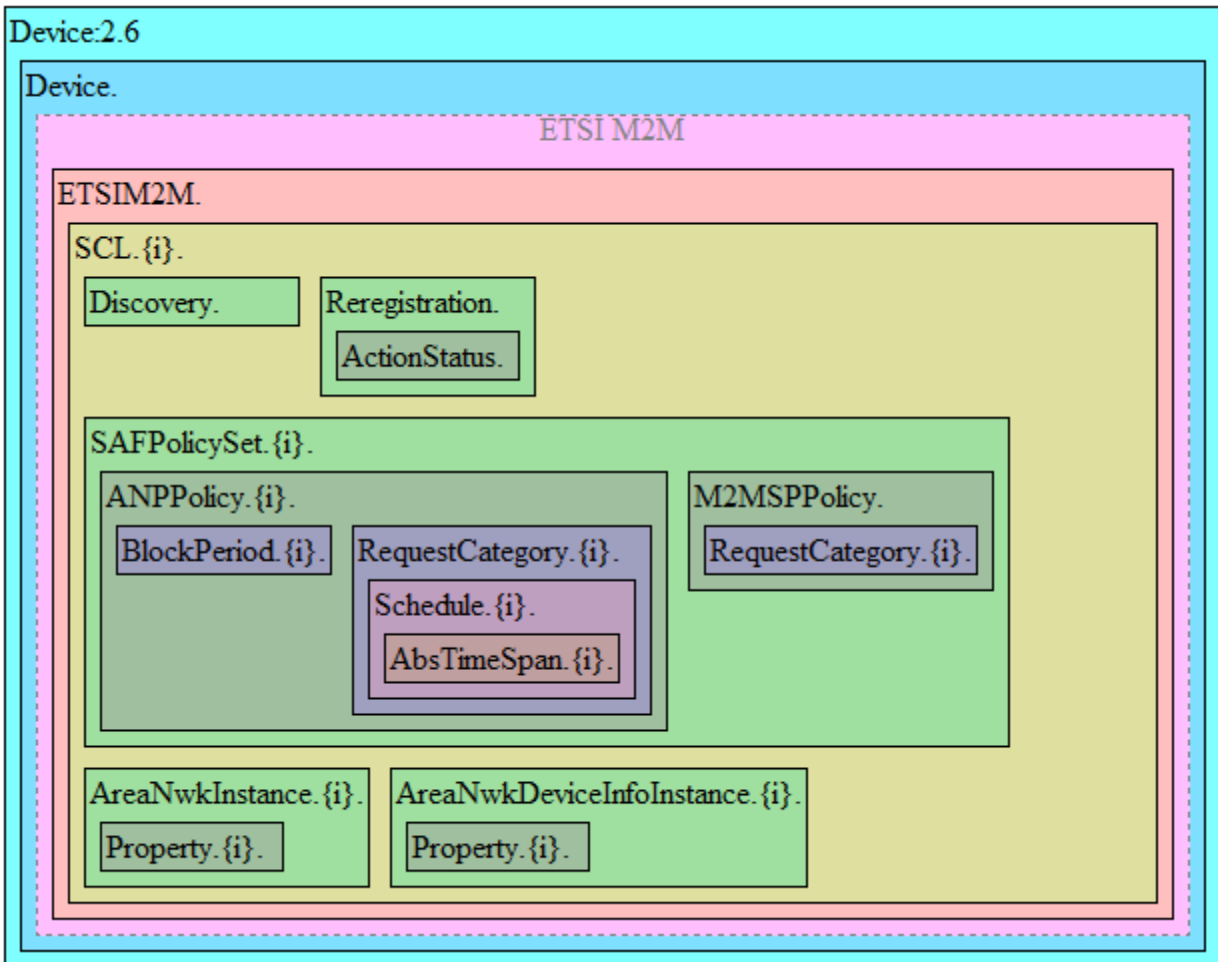



**Figure 47 – ETSI M2M Data Model Structure**

Figure 47 depicts the objects within an ETSI SCL instance.
For deployments where the SCL will only host resources, the following resources must be provisioned:

```
SCL.{1}.
      Enable = true
```

However for deployments where the SCL will transfer messages between M2M Applications and the NSCL, each SCL must have:

An enabled SCL
An enabled default SAFPolicySet
At least 1 enabled ANPPolicy with an enabled Schedule for each of the enabled RequestCategory. There is one enabled RequestCategory instance for each possible RCAT value (e.g., 8 possible values in ETSI release 1.0)
Within the M2MSPPolicy, there is one enabled RequestCategory instance for each possible RCAT value (e.g., 8 possible values in ETSI release 1.0)

As such the following resources must be provisioned:

```
SCL.{1}.
      Enable = true
SCL.{1}.SAFPolicySet.{1}.
      Enable = true
      PolicyScope= default
SCL.{1}.SAFPolicySet.{1}.ANPPolicy.{1}.
      Enable = true
      ANName = AccessNetworkProviderName
SCL.{1}.SAFPolicySet.{1}.ANPPolicy.{1}.RequestCategory.{1}.
      Enable = true
      RCAT = RCAT1
SCL.{1}.SAFPolicySet.{1}.ANPPolicy.{1}.RequestCategory.{1}.Schedule.{1}.
      Enable = true
      Schedules = * * * * *
.
.
SCL.{1}.SAFPolicySet.{1}.ANPPolicy.{1}.RequestCategory.{7}.
      Enable = true
      RCAT = RCAT7
SCL.{1}.SAFPolicySet.{1}.ANPPolicy.{1}.RequestCategory.{7}.Schedule.{1}.
      Enable = true
      Schedules = * * * * *

SCL.{1}.SAFPolicySet.{1}.M2MSPPolicy.RequestCategory.{1}.
      Enable = true
      RCAT = RCAT7
      RankedANList = AccessNetworkProviderName
.
.
SCL.{1}.SAFPolicySet.{1}.M2MSPPolicy.RequestCategory.{7}.
      Enable = true
      RCAT = RCAT7
      RankedANList = AccessNetworkProviderName
```

# Appendix XI Provider Bridge Theory of Operation

A Provider Bridge is defined in 802.1Q-2011 [9] as either a Provider Edge Bridge (PEP) or an S-VLAN Bridge. A PEP provides the capability to stack VLAN tags with the inner tag being the C-TAG and the outer tag being the S-TAG. An S-VLAN Bridge provides a mechanism to process a S-TAG but does not utilize the mechanism to stack C-VLAN tags. The Provider Bridge model supports both of these types of Provider Bridges through the use of the ProviderBridge and VLANTermination objects.

Regarding different traffic bridging rules for Provider Bridges, the possible cases are characterized as follows:

- Provider Edge Bridge as a pure VLAN Bridge
- Stacked VLAN termination in a routed environment
- Internally generated to tagged WAN traffic as a S-VLAN Termination

These scenarios are portrayed in Figure 48, where:

- Residential Domain traffic is treated as a Stacked VLAN termination in a routed environment
- Public Domain and Roaming Domain traffic is treated as a Provider Edge Bridge in a pure VLAN Bridge environment
- Internally generated Device traffic is treated as a S-VLAN termination in a routed environment

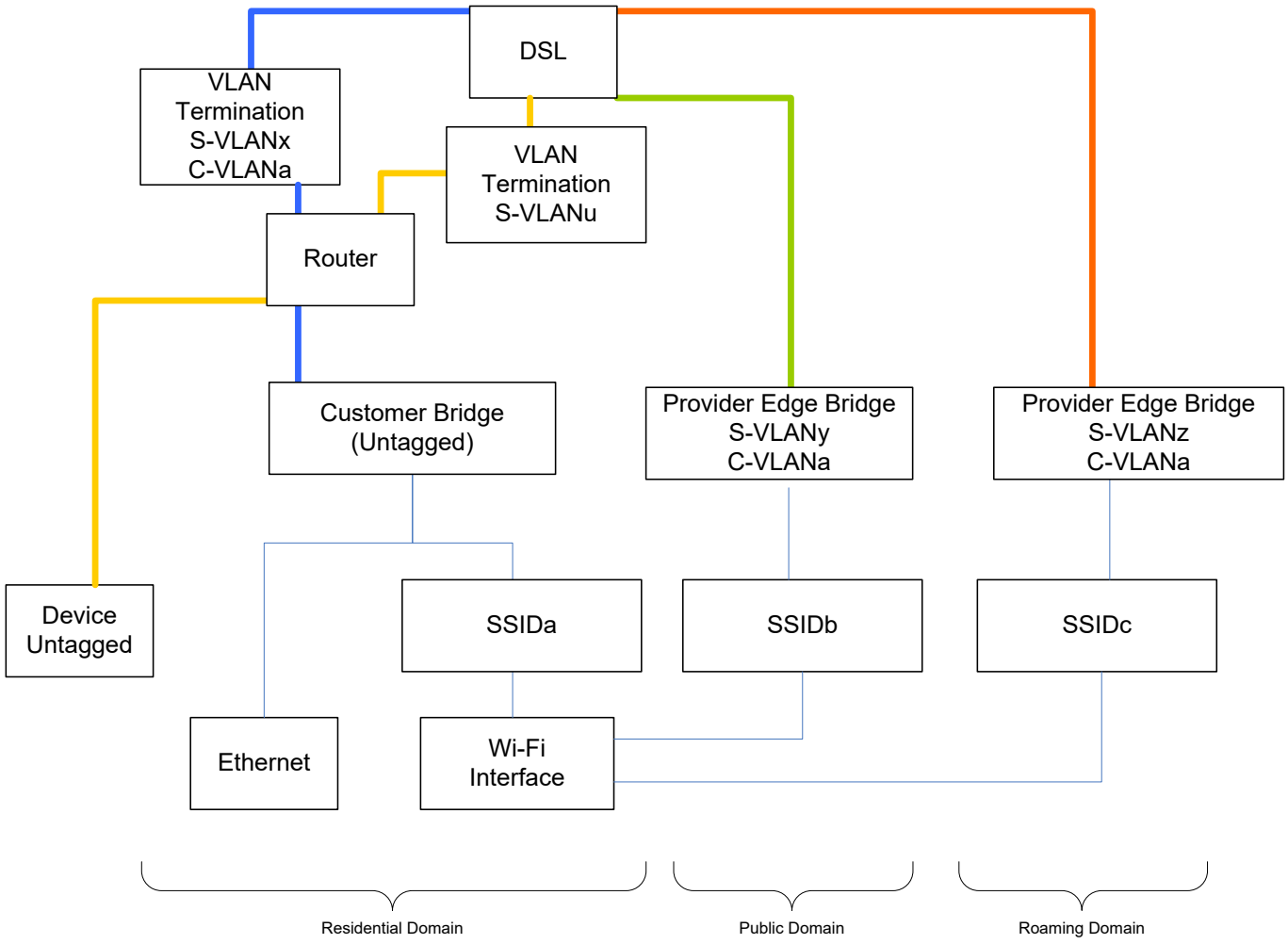



**Figure 48 – Provider Bridge Scenarios**

In order to model the traffic scenarios in Figure 48, the use of the VLANTermination and Bridging Objects are used.

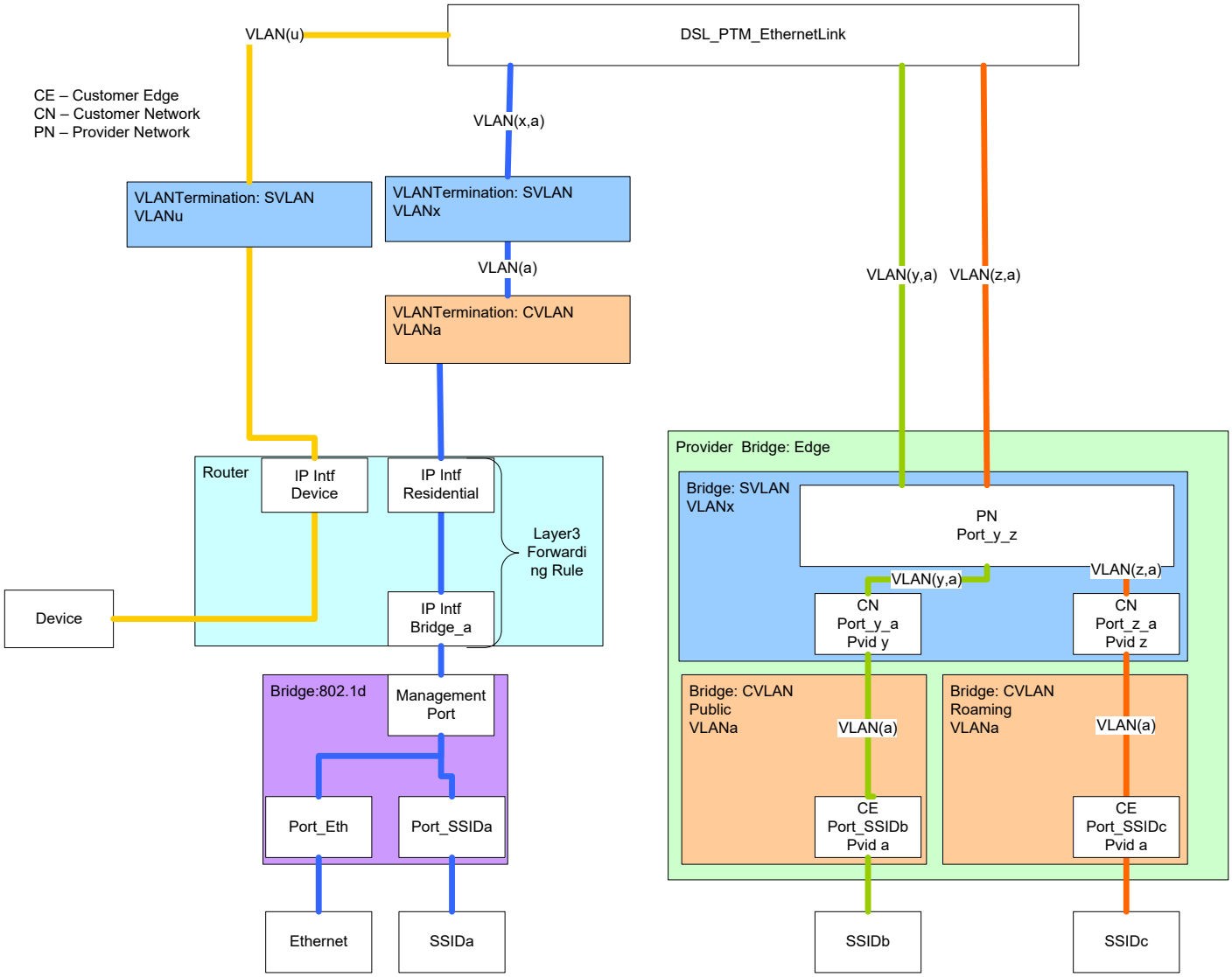



**Figure 49 – Provider Bridge Components**

## XI.1 Residential Domain Scenario

In the Residential Domain scenario untagged traffic is routed from the Ethernet and SSIDa interfaces and tagged with a customer VLAN tag (C-TAG) of VLANa and then double tagged with a Service Provider VLAN tag (S-TAG) of VLANx. This requires the use of:

- 802.1d Bridge instance: This object bridges the residential domain traffic to the Router.
- Layer3 Forwarding Rule: This object ensures that traffic between the Bridge and VLANTermination objects is forwarded to the correct interface. The Rule utilizes the IP Interfaces of the Bridge (IP Intf: Bridge_a) and Residential Domain (IP Intf: Residential)
- VLANTermination object (C-TAG): The C-TAG is applied and removed for traffic egress and ingress to the IP Intf: Residential interface.

- VLANTermination object (S-TAG): The S-TAG is applied and removed for traffic from and to the C-VLAN termination object.

## XI.2 Device Traffic Scenario

In the Device Traffic scenario untagged traffic is routed from the Device and tagged with a Service Provider VLAN tag (S-TAG) of VLANu. This requires the use of:

- VLANTermination object (S-TAG): The S-TAG is applied and removed for traffic egress and ingress to the IP Intf: Device interface.

## XI.3 Public and Roaming Domain Scenarios

In the Public and Roaming Domain scenarios untagged traffic is bridged from the SSIDb and SSIDc interfaces and tagged with a customer VLAN tag (C-TAG) of VLANa and then double tagged with a Service Provider VLAN tag (S-TAG) of VLANy and VLANz respectively. This requires the use of:

- ProviderBridge instance: This object contains and references the customer and service provider bridge components.
- Bridge instance (Customer Public): This object bridges and tags (C-TAG) traffic in the Public Domain to the service provider bridge component.
- Bridge instance (Customer Roaming): This object bridges and tags (C-TAG) traffic in the Roaming Domain to the service provider bridge component.
- Bridge instance (Service Provider): This object add and removes a service provider tag (S-TAG) for customer tagged traffic (C-VLAN) from the Pubic and Roaming Domains.

## XI.4 Provisioning Provider Bridges

A Provider Bridge provides support for Provider Bridges and Provider Edge Bridges as defined in 802.1Q-2011. The difference between a Provider Bridge and a Provider Edge Bridge is that a Provider Edge Bridge incorporates a C-TAG and S-TAG while a Provider Bridge has a S-TAG. The data model differentiates which type of provider using the Type parameter of the ProviderBridge.{i} object.

When configuring the components of a Provider Bridge, the Bridge instance associated with the SVLAN component will have its Device.Bridging.Bridge.{i}.Port.{i}. objects provisioned as either ProviderNetworkPort or a CustomerNetworkPort. Likewise, the CVLAN component(s) will have its Device.Bridging.Bridge.{i}.Port.{i}. objects provisioned as CustomerEdgePorts.

### XI.4.1 Associating Customer Edge Ports with Customer Network Ports

Ports of type CustomerEdgePort are associated with ports of type CustomerNetworkPort by assigning the ports of type CustomerNetworkPort and ports of type CustomerEdgePort to the

port membership (Bridging.Bridge.{i}.VLANPort.{i}.) of the S-VLAN for the Bridge instance of the S-VLAN component.

# Appendix XII ZigBee Theory of Operation

This section explains how the ZigBee Device:2 data model can be used for the management of ZigBee devices.

*NOTE – This Theory of Operation is explained using CWMP but the same principles also apply for USP*

## XII.1 CWMP management using the ZigBee data model

Figure 50 and Figure 51 present the principle and an example basic sequence for the management of ZigBee devices using the Device:2 ZigBee data model. The ZigBee protocol is specified in [42].

The ZigBee devices reside behind a CPE proxy and communicate with the ACS via this CPE proxy. The CPE proxy normally resides in a device such as a broadband router, i.e., a home gateway or an enterprise gateway, and it has a proxy function to translate CWMP messages to ZDO (ZigBee Device Object) function invocations based on the ZigBee data model. The proxy function translates the messages by using a mapping of ZigBee data model objects and CWMP methods to ZDO functions and their parameters. A ZigBee management example using  is shown in Figure 50.

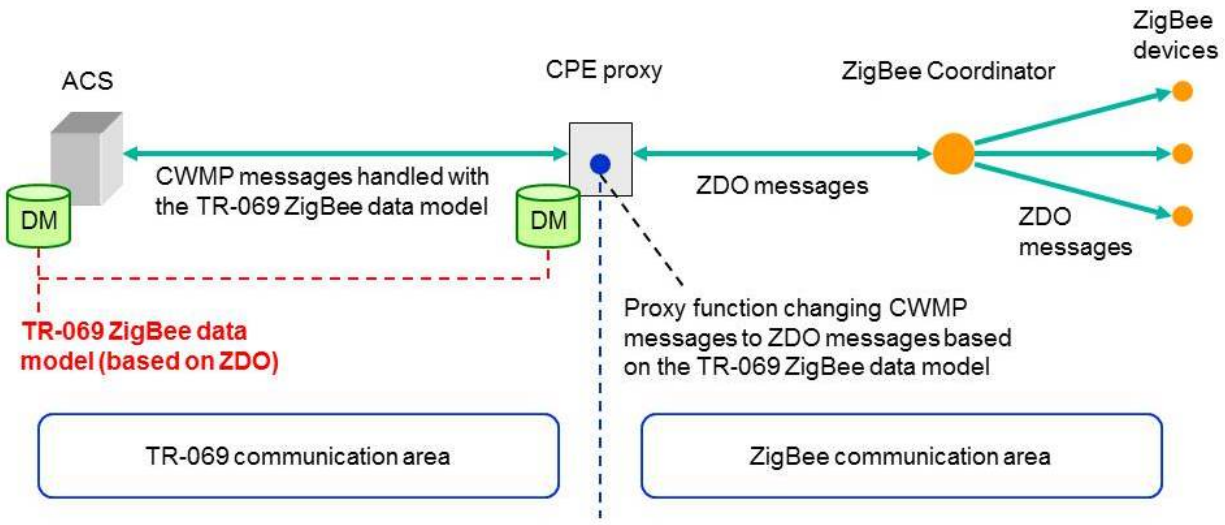



**Figure 50 – Usage of the data model to manage ZigBee devices with TR-069**

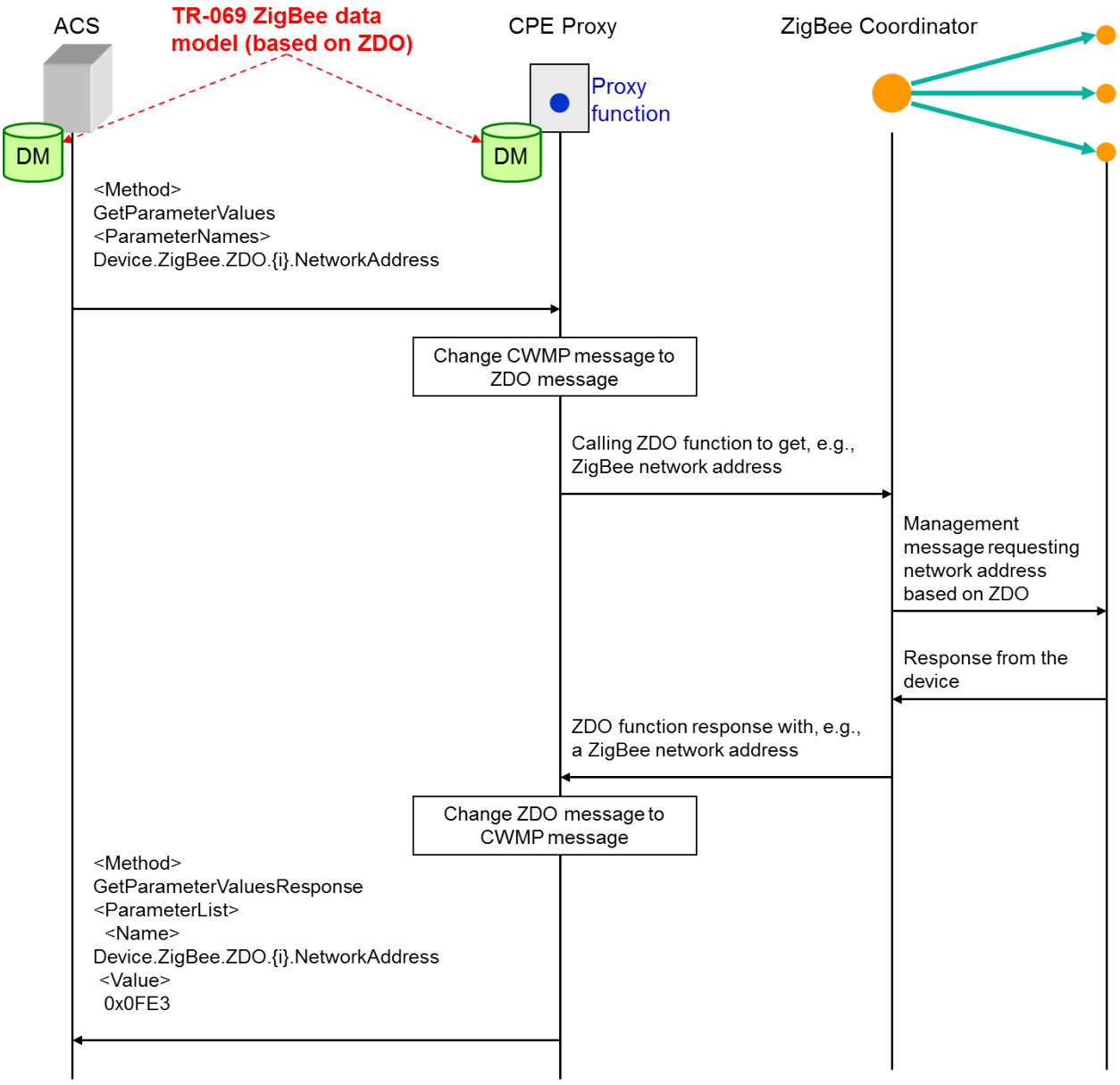



**Figure 51 – Example sequence diagram of ZigBee management with TR-069**

This example shows how the ACS gets the network address of a ZigBee device by using TR-069 communication based on the ZigBee data model. The ACS performs a "GetParameterValues" CWMP method call containing the parameter "Device.ZigBee.ZDO.{i}.NetworkAddress" of the ZigBee data model, which refers to the ZigBee network address. The proxy function in the CPE proxy changes the received message to a ZDO message that calls some ZDO function on the ZigBee Coordinator. The ZigBee Coordinator manages the ZigBee devices according to the called ZDO function and sends the result (the searched network address, in this case) to the proxy. The proxy function changes the ZDO management result to a CWMP message which is denoted in Figure 51 as "GetParameterValuesResponse". The parameter name inside the parameter list is "Device.ZigBee.ZDO.{i}.NetworkAddress" and the corresponding value is "0x0fE3" (network address instance).

## XII.2 CWMP proxying mechanisms and the ZigBee data model

The following two issues related to the proxying of ZigBee devices with the standard TR-069 proxying mechanisms are out of scope of this document:

- Mapping of TR-069 methods plus data model objects/parameters to ZDO functions.
- Description of the exact approaches (and their differences) for proxying a ZigBee device (i) as a Virtual Device or (ii) as an Embedded Device.

However, the following example explains how the main needs of the proxying mechanisms have been taken into account and are covered by the designed data model.

Imagine, for example, a ZigBee coordinator that controls a network which contains, among others, a ZigBee device that is used in a home automation system, i.e., implements the Home Automation Application Profile (0104). Then, the instantiation of the data model for the CPE contains, among others, the following two parameter values (note that "ZC" stands for ZigBee coordinator):

```
Device.ZigBee.ZDO.1.NodeDescriptor.LogicalType = "ZC"
Device.ZigBee.ZDO.2.ApplicationEndpoint.1.ApplicationProfileId = "0104"
```

In order to reference and manage these devices with the EmbeddedDevice mechanism, the CPE instance would simply also include, among others, the following entries:

```
Device.ManagementServer.EmbeddedDevice.1.Reference
```
➔ (*pointing to*) `Device.DeviceInfo.TemperatureStatus.TemperatureSensor.2`
```
Device.ManagementServer.EmbeddedDevice.1.ProtocolReference
```
➔ (*pointing to*) `Device.ZigBee.ZDO.2.ApplicationEndpoint.1`
```
Device.ManagementServer.EmbeddedDevice.2.DiscoveryReference
```
➔ (*pointing to*) `Device.ZigBee.ZDO.1`

For setting the temperature for TemperatureSensor.2, for example, the TR-069 proxy would send a request through the ZigBee coordinator to the Application endpoint referenced by the ProxyReference parameter on the EmbeddedDevice instance. As indicated by the value of `Device.ManagementServer.EmbeddedDevice.1.Reference` in the above example, multiple sensors integrated in the same ZigBee device (i.e., same ZDO instance) can be modeled as different Embedded or Virtual devices while referring to the same ZDO object.

According to the ZigBee protocol, the discovery of ZigBee devices is the responsibility of the ZigBee coordinator. Thus, a ZDO instance that has a LogicalType="ZC" can be made a DiscoveryReference of the various EmbeddedDevice and VirtualDevice instances.

# Appendix XIII Port Control Protocol Theory of Operation

The Port Control Protocol (PCP) allows an IPv6 or IPv4 host to control how incoming IPv6 or IPv4 packets are translated and forwarded by a Network Address Translator (NAT) or simple firewall (generically referred to as the "PCP-controlled device"), and also allows a host to optimize its outgoing NAT keepalive messages.

When a PCP client is embedded in a device, the PCP client can be invoked by:

- Applications running on the device itself (remote access, VoIP…),
- The device GUI,
- The Controller,
- Interworking functions [44] and the PCP proxy that allow applications running on other end-devices connected to the device to manage the PCP-controlled device.

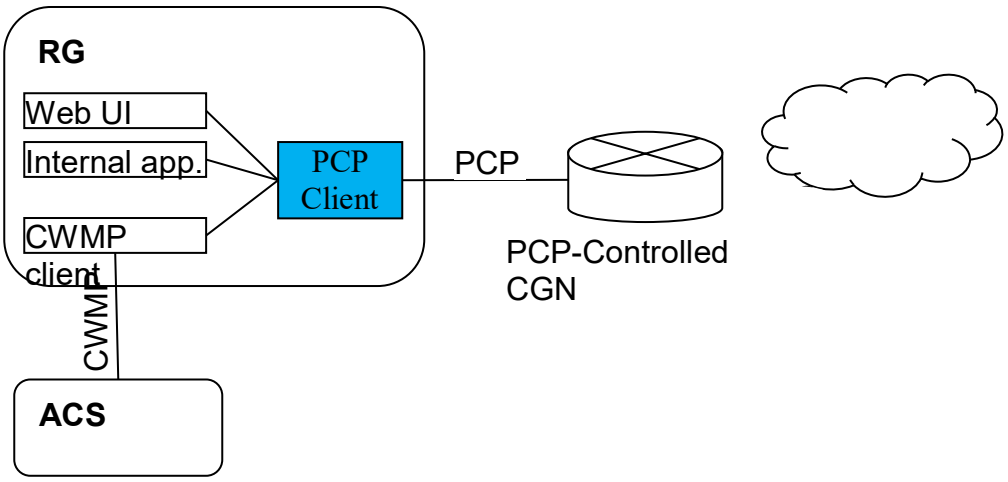



**Figure 52 – Example of a PCP Client embedded in the RG using CWMP**

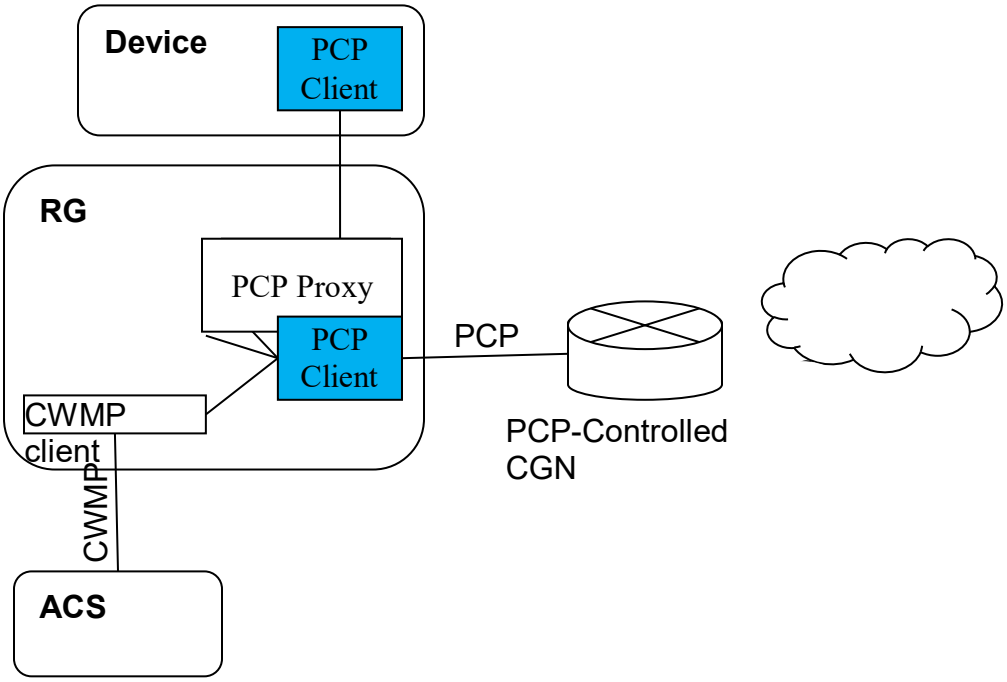



**Figure 53 – Example of a PCP Client embedded in a device using CWMP, with PCP Proxy in the RG**

Defining a PCP data model allows the Controller to remotely manage the PCP client including:

- Configuration and monitoring of the PCP client itself,
- Configuration and monitoring of the PCP servers interacting with the client,
- Monitoring PCP Interworking Functions,
- Monitoring and setting rules in the PCP-controlled device from the PCP client.

Whereas the description of objects themselves is enough to understand how to proceed, some operations need further explanation about the way to manage the objects.

This theory of operation relies on IETF RFCs and drafts:

- RFC 6887 Port Control Protocol (PCP) [43],
- RFC 6970 UPnP IGD-PCP Interworking Function [44],
- DHCP Options for the Port Control Protocol (PCP) [45],
- Port Control Protocol (PCP) Proxy Function [46] ,
- PCP Server Selection [47],
- PCP Flow Examples [48].

The data model allows for more than one PCP client, but those clients operate independently. Therefore, the text below considers only one PCP client.

## XIII.1 Configuration and monitoring of the PCP Server

Prior to sending its first PCP message, the PCP client determines which server to use as described in [47]. To do so the PCP client of the CPE can be configured statically (GUI or CWMP) or via DHCP (v4 or v6).

- When configured via DHCP, the CPE receives a list (at least one) of PCP server addresses in one or more *OPTION_V4_PCP_SERVER* or *OPTION_V6_PCP_SERVER* DHCP options. Based on the content of these DHCP options, the CPE creates one or more instances of *PCP.Client.{i}.Server* (see [45]). The list of addresses provided for each PCP server is stored in the *ServerNameOrAddress* and *AdditionalServerAddresses* parameters and the *Origin* parameter is set to either "DHCPv4" or "DHCPv6".
- When statically configured, one instance of *PCP.Client.{i}.Server* is created per server, with the *Origin* parameter set to "Static". The server is defined by either an FQDN or an IP address in *ServerNameOrAddress*.

Based on these server definitions, the PCP client follows the procedures specified in [47] to determine the IP Address to be used for each configured PCP server.

- While the PCP client is trying to connect to a PCP server on a given IP address, the *PCP.Client.{i}.Server* object's *ServerAddressInUse* holds that IP address and its *Status* is "Connecting".
- When the PCP client has successfully received a response from a server, *Status* becomes "Enabled" and server-discovered properties (*CurrentVersion*, *Capabilities*…) are stored in the corresponding parameters.
- If the PCP client fails to connect to a given PCP server, *ServerAddressInUse* remains the last IP address tried and *Status* reflects the appropriate error condition.

No conflict or doubt can arise between DHCP and static configurations, because they are represented in separate *PCP.Client.{i}.Server* instances, with *Origin* to record the origin of the configuration. *ServerNameOrAddress* is writable by the Controller only if *Origin* is "Static".

## XIII.2 Monitoring and setting rules set by the PCP client

Once a PCP server has been successfully contacted, the PCP client is ready to set rules in the corresponding PCP-controlled device. Depending on the use case, the PCP client selects the appropriate PCP server based on its *Capabilities*, as described in Section 10 of [43]. It is possible to define the following mappings:

Inbound Mapping without filters
An inbound mapping is defined by an instance of the *PCP.Client.{i}.Server.{i}.InboundMapping* table. It is created by a PCP request with the MAP OpCode, as described in Section 11 of [43]. This is allowed only if *PCP.Client.{i}.MAPEnable* is "true".

Inbound Mapping with filters
As above, but additional filters are defined by instances of the *PCP.Client.{i}.Server.{i}.InboundMapping.{i}.Filter* table. Filters are specified in the PCP request using the FILTER option,

as described in Section 13.3 of [43]. This is allowed only if *PCP.Client.{i}.FILTEREnable* is "true".

Outbound Mapping
An outbound mapping is defined by an instance of the *PCP.Client.{i}.Server.{i}.Outbound-Mapping* table. It is created by a PCP request with the PEER OpCode, as described in Section 12 of [43]. This is allowed only if *PCP.Client.{i}.PEEREnable* is "true".

It is possible to define a mapping on behalf of another device. The PCP request uses the THIRD_PARTY option to create the mapping, as described in Section 13.1 of [43]. This is allowed only if *PCP.Client.{i}.THIRDPARTYStatus* is "Enabled".

These operations can be requested by the device itself (embedded applications, GUI, CWMP…) or by another device through the UPnP IGD interworking function [44] (if *PCP.Client.{i}.UPnP-IWF.Status* is "Enabled") or the PCP Proxy [46] (if *PCP.Client.{i}.PCPProxy.Status* is "Enabled").

[48] provides a set of examples to illustrate PCP operations. These operations can be monitored by getting *PCP.Client.{i}.Server.{i}.InboundMapping* and *PCP.Client.{i}.Server.{i}.Outbound-Mapping* objects. The parameters sent by the PCP client in MAP or PEER requests are represented in corresponding parameters *(Lifetime, SuggestedExternalIPAddress, Suggested-ExternalPort, SuggestedExternalPortEndRange, ProtocolNumber, InternalPort...)* of *PCP.Client.Server.{i}.InboundMapping* and *PCP.Client.Server.{i}.OutboundMapping*. The *Origin* parameter denotes which mechanism triggered the request:

- "Internal" for an embedded application,
- "Static" for a request issued from the GUI or set using CWMP (see next paragraph),
- "UPnP_IWF" for a UPnP IGD device,
- "PCP_Proxy" for a PCP device.

The parameters received when the PCP-controlled device has processed the request are represented in corresponding parameters *(Lifetime, AssignedExternalIPAddress, AssignedExternalPort, AssignedExternalPortEndRange...)* of *PCP.Client.{i}.Server.{i}.In-boundMapping* and *PCP.Client.{i}.Server.{i}.OutboundMapping*.

To remotely create rules using CWMP or USP, the Controller configures the request to be sent by the PCP Client. To do so the Controller creates the necessary objects and sets, depending on the operation, the *Lifetime, SuggestedExternalIPAddress, SuggestedExternalPort, SuggestedExternalPortEndRange, ProtocolNumber, InternalPort* parameters of *PCP.Client.{i}.Server.{i}.InboundMapping* or of *PCP.Client.{i}.Server.{i}.OutboundMapping*. To monitor the result, the Controller will get *PCP.Client.{i}.Server.{i}.InboundMapping* and *PCP.Client.{i}.Server.{i}.OutboundMapping* objects to retrieve the parameters received from the PCP-controlled device.

## XIII.3 Rapid recovery

A recovery mechanism for situations where the PCP server loses its state is described in Section 14 of [43]. This is usable only if *PCP.Client.{i}.ANNOUNCEEnable* is "true".

# Appendix XIV GRE Tunnel Theory of Operation

See Annex B for general information on how tunneling is modeled.

RFC 2784 [50] defines a generic mechanism to encapsulate a packet of protocol A (known as the payload protocol) in a GRE packet. The resulting GRE packet is then encapsulated into a protocol B (known as the delivery protocol). The result of this operation is a payload packet that is encapsulated in a GRE tunnel delivered via protocol B. RFC 2890 [51] extends the GRE header with two optional fields. The Key field provides an identifier to identify flows within the GRE tunnel. The Sequence Number field is used to maintain the sequence of packets within the GRE tunnel.

Device:2 models a GRE tunnel using the *GRE.Tunnel* object. Multiple GRE flows to the same remote endpoint are possible by defining multiple *GRE.Tunnel.{i}.Interface* instances within the same *GRE.Tunnel* instance.

This Appendix describes the usage of GRE for two scenarios: L2 payload over GRE and L3 payload over GRE.

## XIV.1 L2 Payload over GRE

For this example consider a Provider Edge Bridge that discriminates 2 separate VLANs as shown in Figure 54. In this case the service provider does not support a VLAN infrastructure at the access node, but does at the core network.

A GRE tunnel is used to preserve the VLAN tagging at the edge to further interconnect the other VLAN segments. In this scenario, as the remote endpoint is the same in both cases, the VLANs are modeled as two flows within a single instance of the *GRE.Tunnel.{i}* object.

In addition, the *DSCPMarkPolicy* parameter can be used to assign DSCP values to each *GRE.-Tunnel.{i}.Interface* instance for QoS treatment in the access network and towards the GRE concentrator.

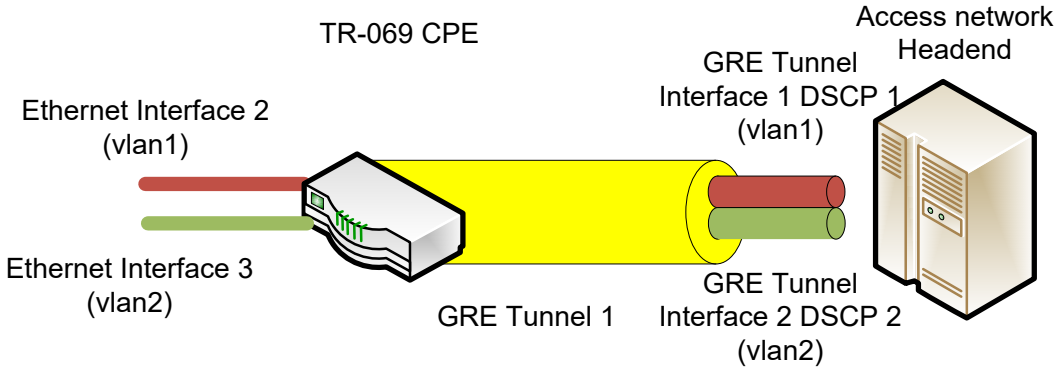



**Figure 54 – VLAN Traffic over GRE**

The GRE Tunnel interface layout is shown in Figure 55.

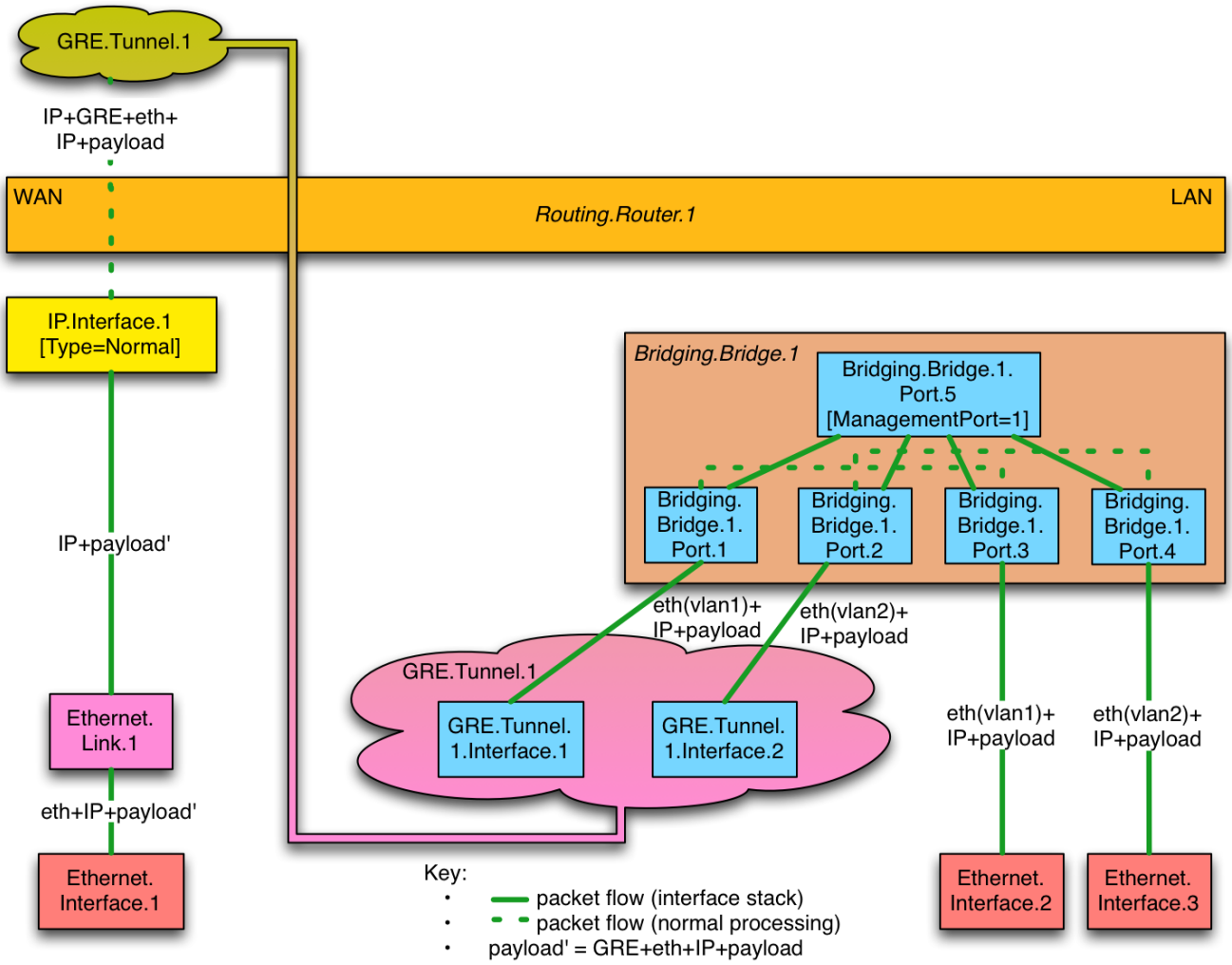



**Figure 55 – L2 over GRE Tunnel**

The configuration for this scenario assumes that the WAN Ethernet interface, Ethernet Link and IP interface objects have been previously configured; likewise the LAN Ethernet and Bridging objects have been previously configured. This section focuses on the association and configuration of the GRE tunnel with the WAN IP interface and the Bridge Ports.

The example configuration uses the RFC 2890 [51] Key field to determine the GRE tunnel interface to which the GRE tunnel will forward packets.

***GRE Tunnel***

```
Device.GRE.Tunnel.1.Enable = True
Device.GRE.Tunnel.1.RemoteEndPoints = GRE-IPAddress
Device.GRE.Tunnel.1.DeliveryHeaderProtocol = IPv4
```

***GRE Tunnel Interface 1***

```
Device.GRE.Tunnel.1.Interface.1
Device.GRE.Tunnel.1.Interface.1.Enable = True
Device.GRE.Tunnel.1.Interface.1.KeyIdentifierGenerationPolicy = Provisioned
```

```
Device.GRE.Tunnel.1.Interface.1.KeyIdentifier = 1

GRE Tunnel Interface 2
Device.GRE.Tunnel.1.Interface.2
Device.GRE.Tunnel.1.Interface.2.Enable = True
Device.GRE.Tunnel.1.Interface.2.KeyIdentifierGenerationPolicy = Provisioned
Device.GRE.Tunnel.1.Interface.2.KeyIdentifier = 2

Associate Bridge Ports with GRE Tunnel Interfaces
Device.Bridging.Bridge.1.Port.1.LowerLayers = Device.GRE.Tunnel.1.Interface.1
Device.Bridging.Bridge.1.Port.2.LowerLayers = Device.GRE.Tunnel.1.Interface.2

Assign the DSCP value to each GRE Tunnel Interface using the GRE.Filter
Device.GRE.Filter.1
Device.GRE.Filter.1.Enable = True
Device.GRE.Filter.1.Order = 1
Device.GRE.Filter.1.Interface = Device.GRE.Tunnel.1.Interface.1
Device.GRE.Filter.1.DSCPMarkPolicy = DSCP1

Device.GRE.Filter.2
Device.GRE.Filter.2.Enable = True
Device.GRE.Filter.2.Order = 2
Device.GRE.Filter.2.Interface = Device.GRE.Tunnel.1.Interface.2
Device.GRE.Filter.2.DSCPMarkPolicy = DSCP2
```

## XIV.2 L3 Payload over GRE

This example describes an IP in IP encapsulation where a GRE tunnel takes IPv4 payload and encapsulates over IPv6.

Figure 56 shows the scenario where an IPv4 LAN network is tunneled in an IPv6 GRE tunnel that uses IPv6 global addresses.

The GRE tunnels use the default IPv6 WAN interface of the CPE.

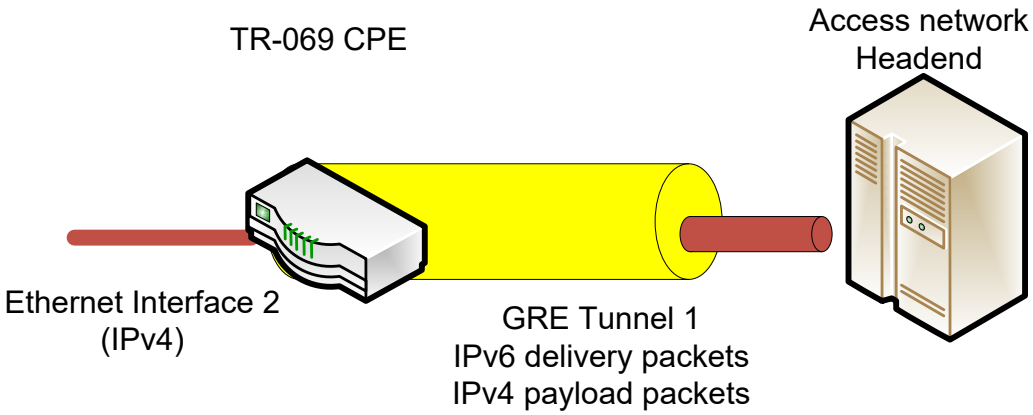



**Figure 56 – IP over IP GRE Encapsulation**

Figure 57 shows the configuration of a GRE tunnel for an IPv4 Private network attached to a LAN interface that is encapsulated in the IPv6 packet.

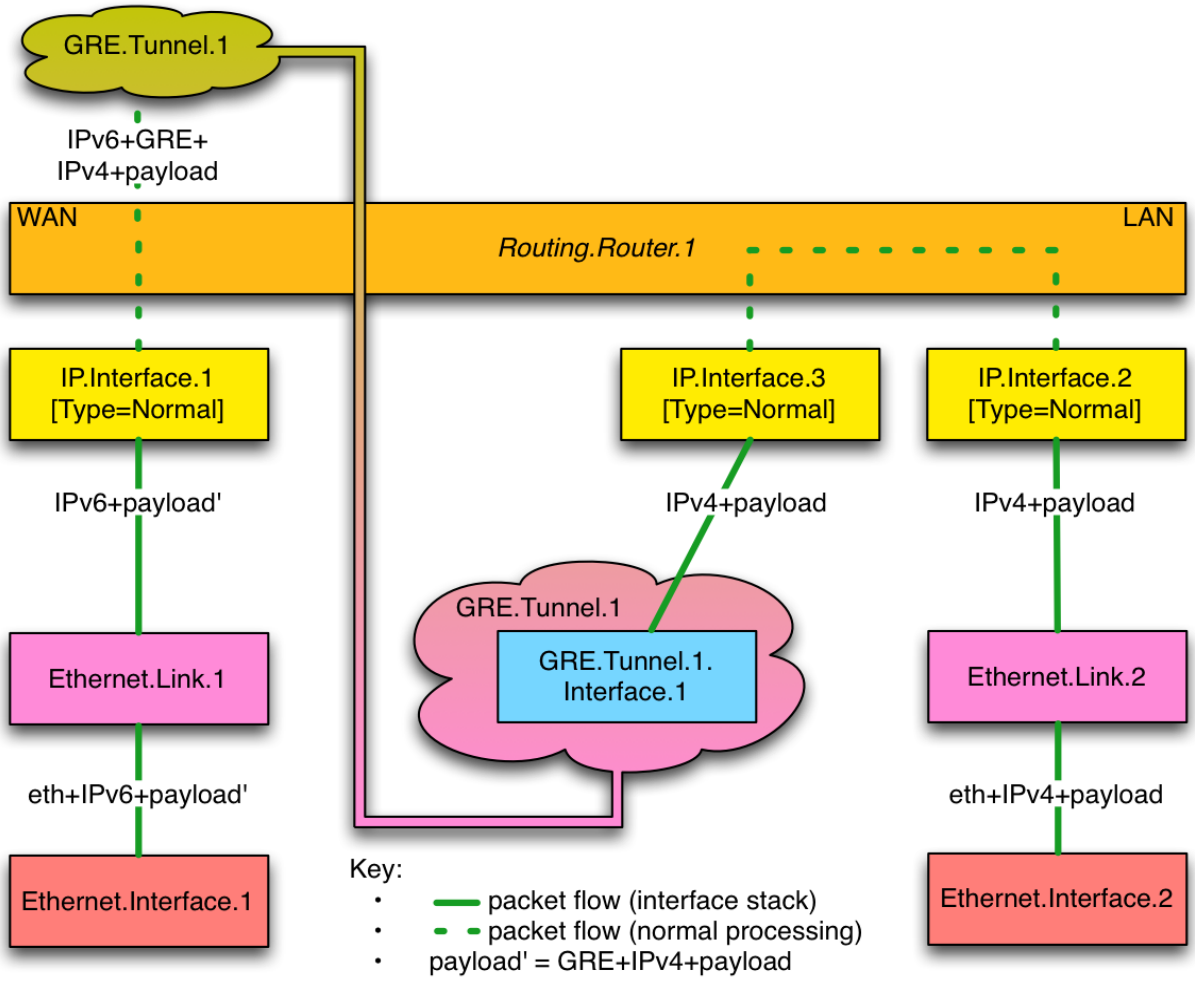



**Figure 57 – L3 over GRE Tunnel**

The configuration for this scenario assumes that the WAN and LAN Ethernet interface, Ethernet Link and IP interface objects have been previously configured. This section focuses on the association and configuration of the GRE tunnel with the WAN and Tunnel IP interfaces.

***GRE Tunnel***

```
Device.GRE.Tunnel.1.Enable = True
Device.GRE.Tunnel.1.RemoteEndPoints = GRE-IPAddress
Device.GRE.Tunnel.1.DeliveryHeaderProtocol = IPv6
```

***GRE Tunnel Interface 1***

```
Device.GRE.Tunnel.1.Interface.1
Device.GRE.Tunnel.1.Interface.1.Enable = True
```

***Associate Tunnel IPv4 Interface with GRE Tunnel Interface***

```
Device.IP.Interface.3.LowerLayers = Device.GRE.Tunnel.1.Interface.1
```

# Appendix XV MAP Theory of Operation

See Annex B for general information on how tunneling is modeled.

MAP (Mapping of Address and Port) is a mechanism for transporting IPv4 packets across an IPv6 network, and a generic mechanism for mapping between IPv6 addresses and IPv4 addresses and ports. There are two mutually exclusive MAP transport modes, both of which use NAPT44 (modified to use a restricted port range):

- MAP-E (Encapsulation) [52] uses an IPv4-in-IPv6 tunnel.
- MAP-T (Translation) [54] uses stateless NAT64.

Many aspects of the MAP configuration are the same for both MAP-E and MAP-T. [53] defines DHCPv6 options for configuring MAP parameters, and the Device:2 data model parameters correspond closely to these parameters.

## XV.1 MAP Configuration Parameters

The MAP-T architecture is illustrated in Figure 58. The MAP-E architecture diagram looks very similar, but differs as follows:

- The CPE's MAP function involves 6-4 encapsulation rather than 6-4 translation.
- The CPE uses a Border Router (BR) IPv6 address rather than a prefix.
- Non MAP-aware servers (i.e. native IPv6 servers) can't be reached by IPv4 devices behind the CPE (i.e. can't be part of the MAP domain).

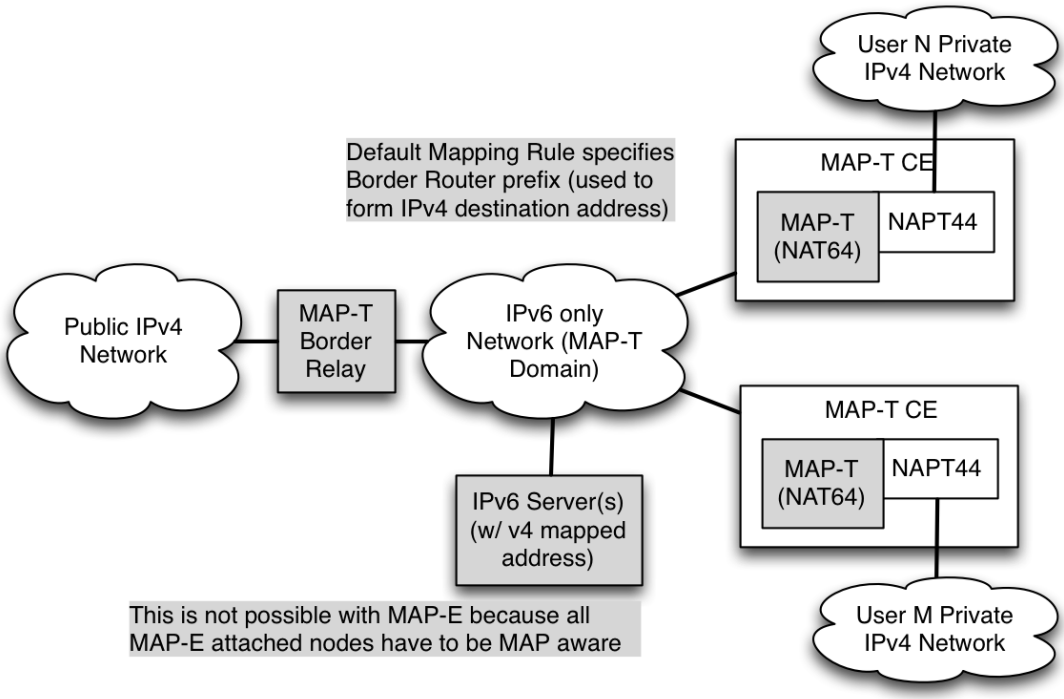



**Figure 58 – MAP-T Architecture**

The Device:2 data model models each MAP domain as an instance of the corresponding *MAP.Domain* table. The most important domain parameters are:

- *TransportMode*: "Encapsulation" (MAP-E) or "Translation" (MAP-T).
- *WANInterface*: the WAN IP interface through which all MAP traffic will flow.
- *IPv6Prefix*: end-user IPv6 prefix; one of this interface's prefixes, typically assigned via DHCPv6 Prefix Delegation.
- *BRIPv6Prefix*: the Border Router IPv6 prefix (MAP-T mode) or IPv6 address (MAP-E mode).
- *DSCPMarkPolicy*: governs DSCP selection when encapsulating / translating.
- *PSIDOffset* etc: parameters defining Port-sets ([52] Section 5.1).

Each domain has a set of mapping rules ([52] Section 5) with each rule having the following parameters:

- *IPv6Prefix*: the IPv6 prefix for this rule.
- *IPv4Prefix*: the IPv4 prefix for this rule.
- *EABitsLength*: the length of the EA (Embedded Address) bits for this rule.
- *IsFMR*: whether this rule is an FMR (Forwarding Mapping Rule).

The mapping rule with the longest match between its *IPv6Prefix* and the end-user IPv6 prefix is the BMR (Basic Mapping Rule). This is used to determine the MAP IPv6 address, which is one of *Interface*'s addresses and is used for all MAP traffic.

## XV.2 Internal Treatment of IPv4 Packets

Since a device can have multiple upstream and multiple downstream interfaces, the model supports a logical representation of the internal virtual MAP IPv4 interface according to the general pattern described in Annex B. The *IPv4Forwarding* entries will route traffic between the LAN IPv4 interface and the MAP IPv4 interface.

Figure 59 shows the flow of MAP traffic through the various interfaces. Noted in the figure are sample values for the various *IP.Interface* entries that would be needed.

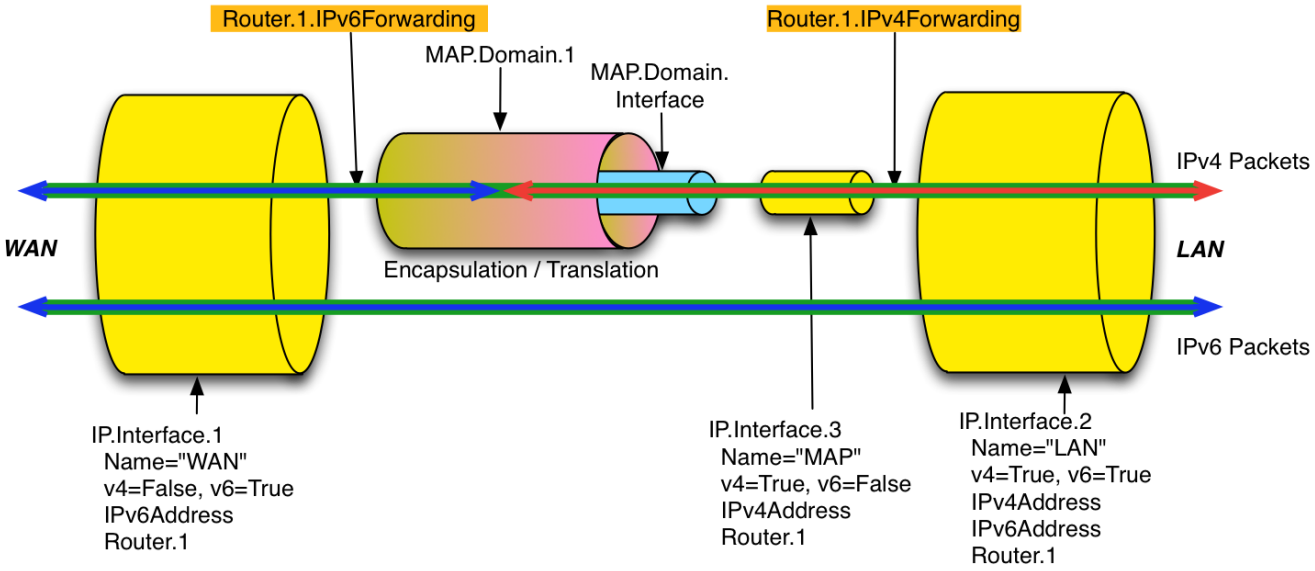



**Figure 59 – Sample MAP Routing and Forwarding**

Figure 60 shows the corresponding MAP interface stack.

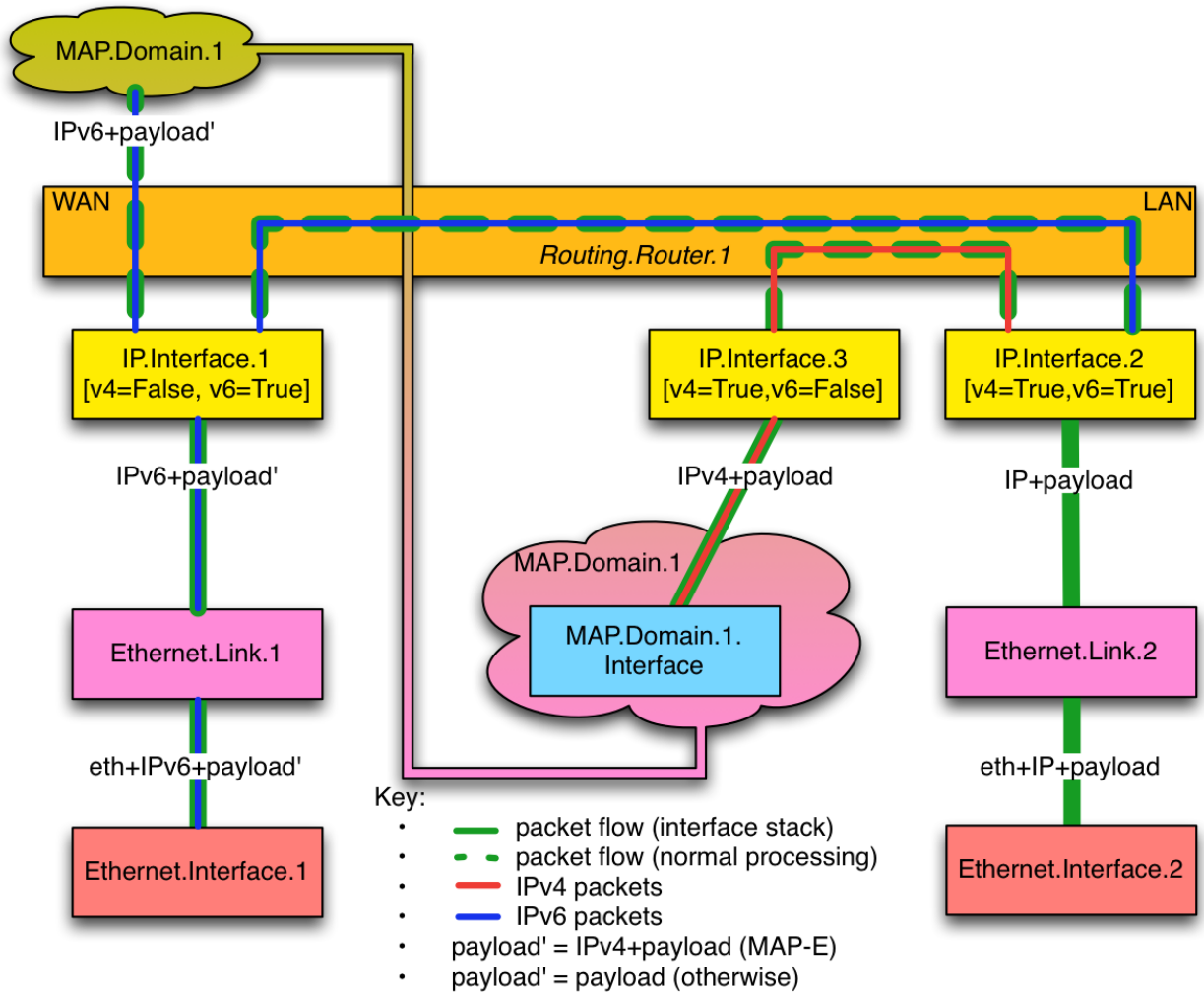



**Figure 60 – Sample MAP Routing and Forwarding (Interface Stack)**

# Appendix XVI G.fast Theory of Operation

G.fast (hereafter referred to as FAST) is a DSL communications technology defined by ITU-T G.9700, G.9701, and G.997.2.

Devices that support both DSL and FAST (both interfaces' objects are administratively Enabled) have the capability to switch from one mode to another. If the device is connected in xDSL mode (DSL.Line.{i}.status is "Up"), FAST interface is down (FAST.Line.{i}.status is "Not Present" or "Down"). The InterfaceStack Table needs to reflect the relationship between the PTM interface and DSL interface as seen in Figure 61. The PTM's LowerLayers points to DSL.Channel instance whose status is "Up".

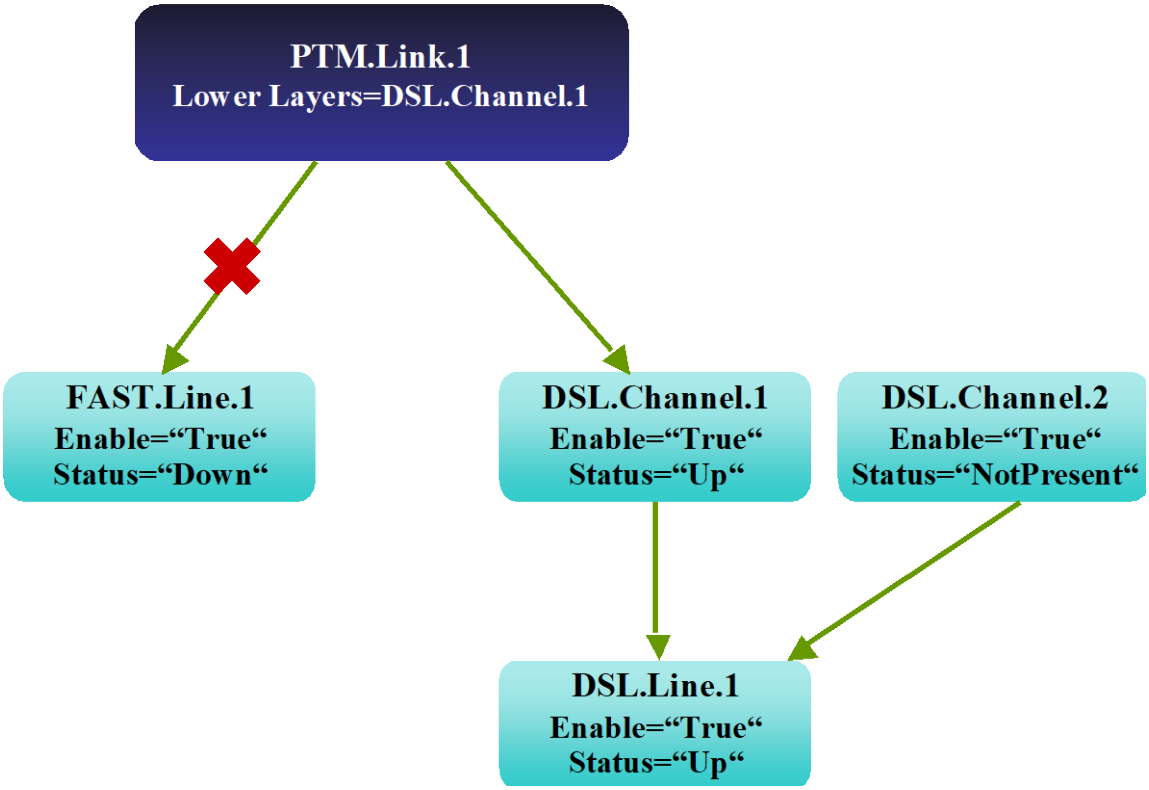



Figure 61 – PTM Link for DSL mode Line

In the case when the device is connected in FAST mode, the DSL line is down. The InterfaceStack Table needs to show that the PTM's LowerLayers points to the FAST.Line interface as below:

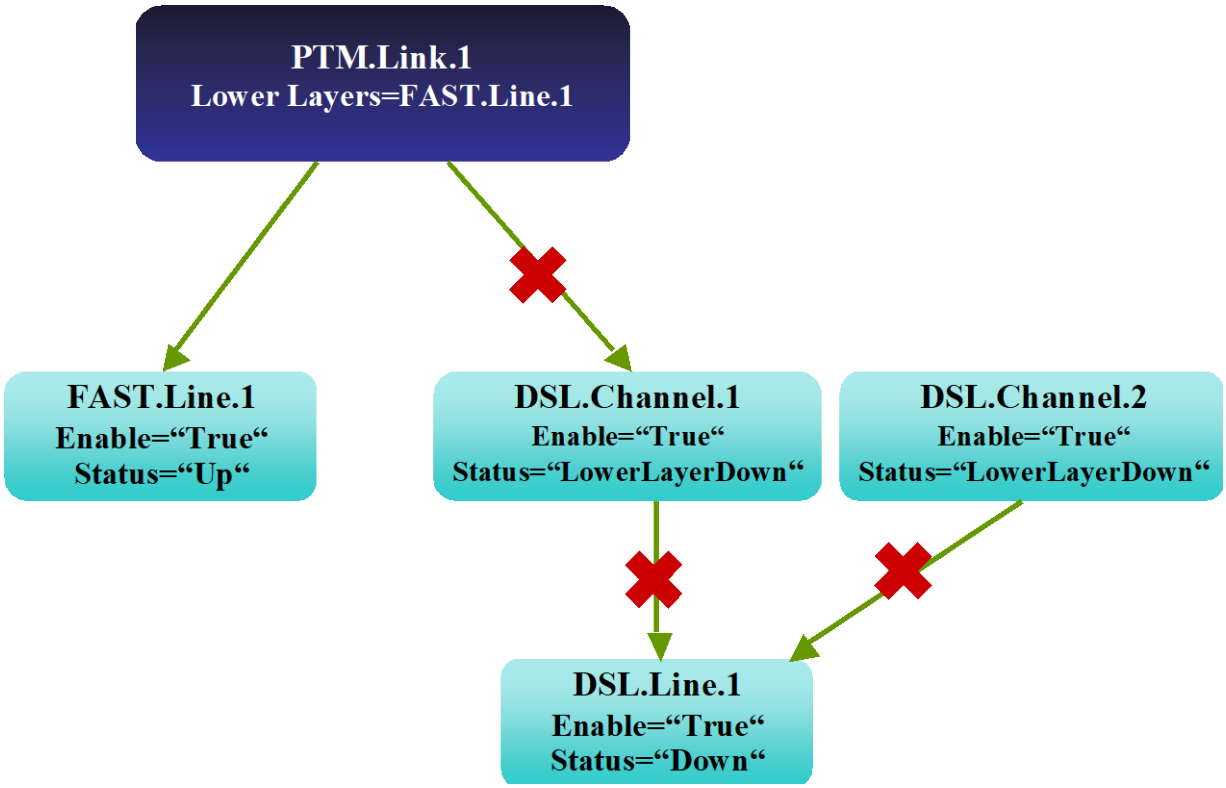



Figure 62 – PTM Link for FAST mode Line

The same fall back mechanism applies to the bonding of FAST and DSL interfaces. PTM's interface is to be stacked on two bonding groups as they are both administrative "Enable". However, in the InterfaceStack Table, the PTM interface's LowerLayers points to the bonding group that has Operational Status "Up". In the diagram below, PTM's LowerLayers points to the bonding group of FAST.Line, which is currently "Up". The DSL bonding group instance corresponding to DSL channels is "Down".

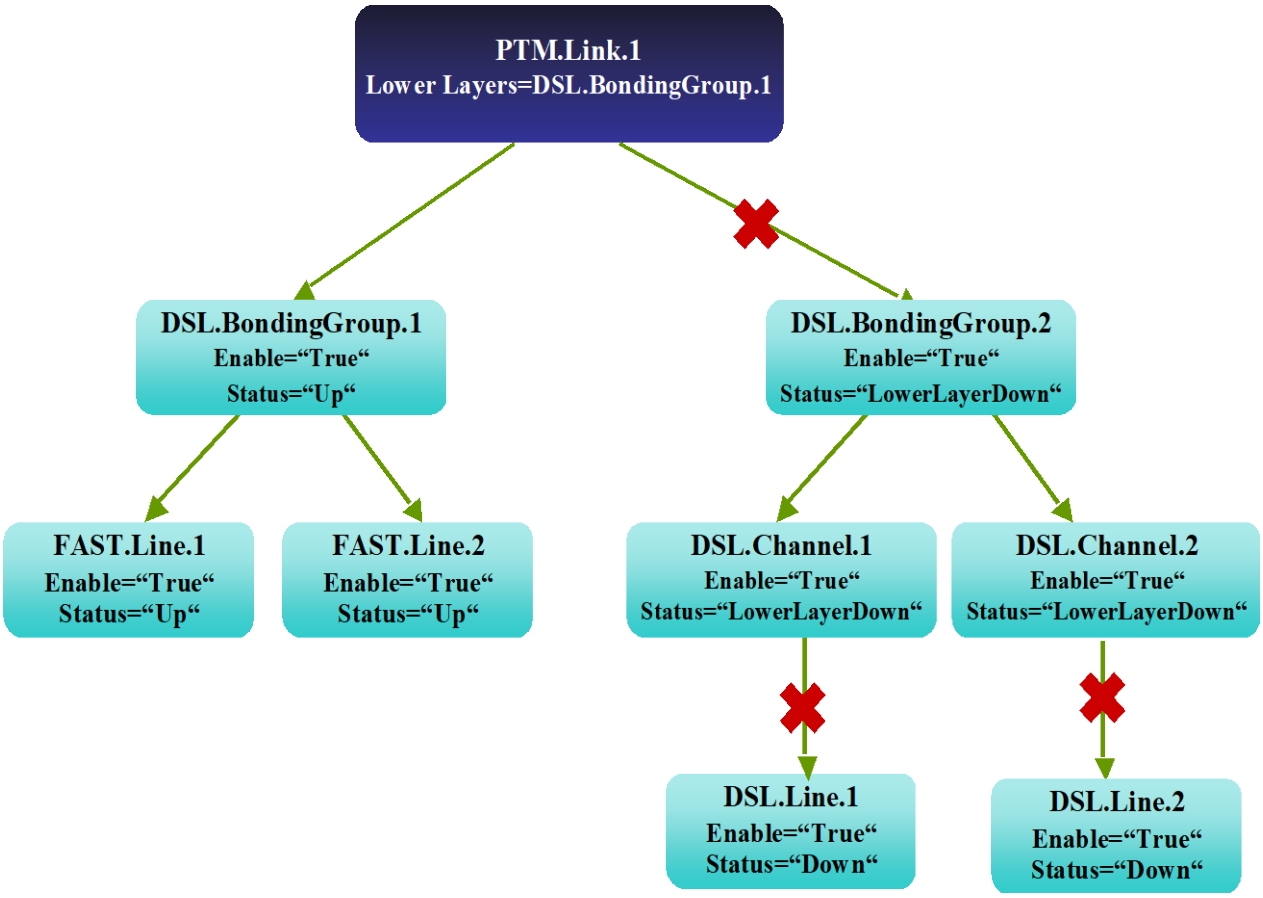



Figure 63 – PTM Link Bonding Groups for FAST mode Lines

In the case where DSL Bonding group is "Up" for non-FAST mode lines, the diagram below shows PTM's LowerLayers pointing to the bonding group of DSL.Channel, which is currently "Up". The DSL bonding group instance corresponding to FAST Lines is "Down" here.

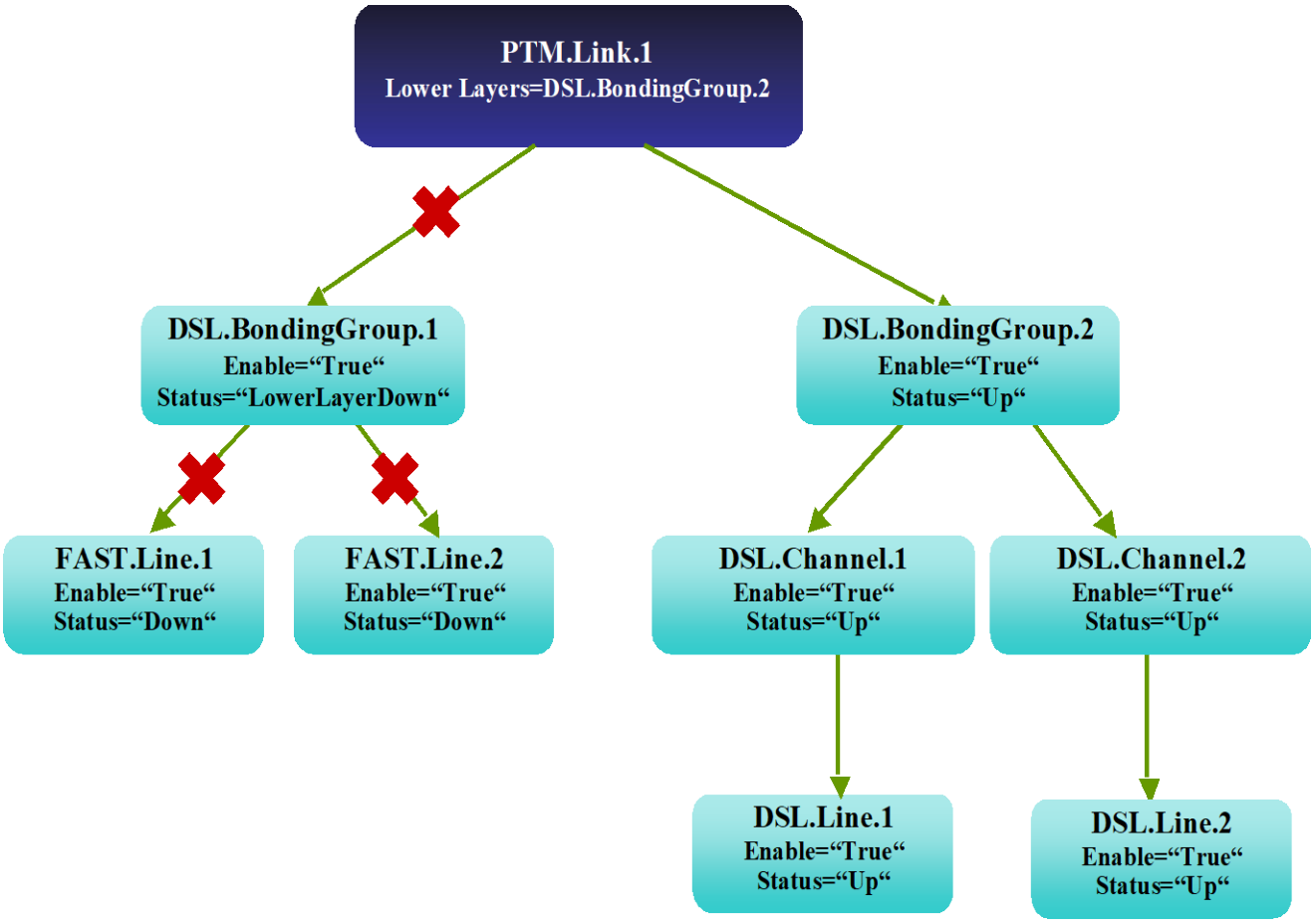



**Figure 64 – PTM Link Bonding Groups for DSL mode Lines**

# Appendix XVII USB Host Theory of Operation

## XVII.1 Overview

An increasing number of devices are equipped with a USB Host controller and USB host interface(s) / connector(s).

There are a number of use cases for adding a USB Host and connected devices to a CWMP data model. One example is retrieving the exact product identity of the connected device in the event of service issues such as printer or file sharing problems. Another example is notifying the user that a newly-connected device is not supported, e.g. due to a missing driver. Or the detection of the connection of a particular USB device could mean additional services for this device could be offered to the subscriber.

The data model contains the number of devices connected to each host controller. For each device, the main properties of the USB device descriptors as well as interface descriptors are represented. The latter is to support devices that only indicate class/subclass (and therefore device type) at the interface level.

Example USB topology of connected devices:

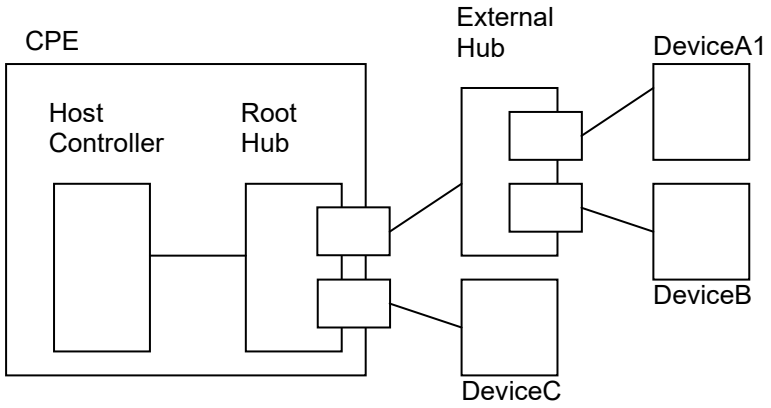



**Figure 65 - Example USB Host Connections**

All USB devices attach to a USB Host through a port on a USB entity known as a hub. Hubs have status bits that are used to report the attachment or removal of a USB device on one of its ports. The USB Host queries the hub to retrieve these status bits. In the case of an attachment, the USB Host enables the port and addresses the USB device through the device's control pipe at the default address. Figure 65 depicts both a Root Hub and an External Hub that provide this service.

The USB Host assigns a unique USB address to the device and then determines if the newly attached USB device is a hub or function. The USB Host establishes its end of the control pipe for the USB using the assigned USB address and endpoint number zero. This is reflected in the data model by adding a new USBHosts.Host.{i}.Device.{i}. instance.

If the attached USB device is a hub and USB devices are attached to its ports, then the above procedure is followed for each of the attached USB devices.

If the attached USB device is a function, then attachment notifications will be handled by the USB Host software that is appropriate for the function.

# Appendix XVIII Location Management

This section discusses the Theory of Operation for Location Management using CWMP [2] or USP [67] and the Location object defined in the <rootobject>.DeviceInfo data model.

## XVIII.1 Overview

The Location object defined in this document is a multi-instance object that can be used by any device that needs to be able to acquire and/or express its "location."
This Location object is a multi instance object to account for the fact that a Device can acquire location information in more than one way. Location info can be acquired by:

- GPS/A-GPS, i.e. provided by specific on-board circuitry such as GPS or A-GPS;
- Manual, i.e. manually configured via the Device local GUI
- External, i.e. remotely configured via a number of protocols, including e.g. TR-069

Location objects can be created autonomously by the device, based on the location information it receives by CWMP or USP. When the Location object is created autonomously by the device, the device itself will fill the DataObject parameter with location data coming from GPS/AGPS, local GUI or an external protocol (not CWMP). When created by CWMP or USP, it is up to the CWMP or USP protocol to configure the DataObject parameter. Regardless of how a Location object is created, the device is responsible for populating the values of all of the location metadata (i.e., all parameters except the DataObject that contains the location information and the AcquiredTime) not writable by any external mechanism.
When a Location object is updated, the object can only be updated through the same mechanism that created it. The device will update the AcquiredTime as necessary and place the updated location data in the DataObject.
When a Location object is deleted, the object can only be deleted through the same mechanism that created it.

## XVIII.2 Multiple Instances of Location Data

Devices that need to make use of location data will need to have rules around how to deal with multiple instances of location data. These rules are out of scope for CWMP or USP and the Device:2 data model. These rules may need to be specific to a particular application. For example, if a VoIP device chooses to send location data in a SIP message, the device can include all of the instances of DataObject in that message, order the Locations Objects according to the acquisition date and time (parameter AcquiredDateTime, most recent is first) or order the Location objects according to some sort of protocol preference, such as GPS, A-GPS, DHCP, HELD, CWMP, USP, and then all others according to acquisition date and time.

A Femtocell Access Point (FAP) with multiple sources of location can also need rules for use of the Location object. If it must make decisions locally based on location, the FAP will need rules to determine the preferred location. If the FAP must send its location elsewhere, the FAP will

need rules to determine whether the FAP sends all of its location data, or selects certain locations based on specific criteria.

## XVIII.3 CWMP, USP, Manual, GPS, and AGPS Configured Location

As noted in the description of the Device:2 data model parameter <rootObject>.Location.{i}.DataObject., Manual, GPS, and AGPS mechanisms are formatted by the device according to the following formats specified by the IETF. A Controller that is creating an External:CWMP or an External:USP location will use one of these formats:

1. Geographical coordinates formatted according to the XML syntax specified in IETF RFC5491[57] (update of RFC4119[56])
2. Civic addresses according to the XML syntax specified in IETF RFC5139[58] (update of RFC4119 [56] )

Location information in these IETF RFCs is specified within the IETF framework of presence information. While these IETF RFCs specify presence information different from the Location component model assumed in the TR-069 framework, the IETF data format is adopted by BBF independent of these higher level differences.

IETF defines its XML syntax for geographical information as a subset of presence information (<presence> object in the XML example below), generally related to a device (<device> object) or a user (<user> object). IETF location information is represented using a Presence Information Data Format Location Object (PIDF-LO). This is represented as the <geopriv> object in the XML example below.

### XVIII.3.1 Example: Manual, GPS, AGPS, and External:CWMP <rootObject>.Location.{i}.DataObject. Format

This example, modified from an example in RFC5491, explains how to format location information in a <rootObject>.Location.{i}.DataObject. parameter with both geographical coordinates and civic location information according to the above-referenced IETF RFCs. The schema associated with the civic location namespace "urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr" is specified in RFC5139[58].

```
<presence xmlns="urn:ietf:params:xml:ns:pidf"
xmlns:dm="urn:ietf:params:xml:ns:pidf:data-model"
xmlns:gp="urn:ietf:params:xml:ns:pidf:geopriv10"
xmlns:gml="http://www.opengis.net/gml"
xmlns:cl="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr"
entity=" ">
      <dm:device id=" FFFFFF-FAP-123456789 ">
            <gp:geopriv>
                  <gp:location-info>
                        <gml:Point srsName="urn:ogc:def:crs:EPSG::4326">
                              <gml:pos>-43.5723 153.21760</gml:pos>
                        </gml:Point>
                        <cl:civicAddress>
```

```
                                <cl:FLR>2</cl:FLR>
                            </cl:civicAddress>
                        </gp:location-info>
                        <gp:usage-rules/>
                        <gp:method>Wiremap</gp:method>
                    </gp:geopriv>
                    <dm:deviceID>mac:8asd7d7d70</dm:deviceID>
                    <dm:timestamp>2007-06-22T20:57:29Z</dm:timestamp>
            </dm:device>
</presence>
```

### XVIII.3.2 RFC5491 and RFC5139 Location Element Definitions

The XML elements are defined as follows by the IETF in RFC5491 [57] and related documents:

1. <presence> (RFC5491 [57])
   The <presence> element MUST have an 'entity' attribute. The value of the 'entity' attribute is the 'pres' URL of the presentity publishing this presence document. The<presence> element MUST contain a namespace declaration ('xmlns') to indicate the namespace on which the presence document is based. The presence document compliant to this specification MUST have the namespace 'urn:ietf:params:xml:ns:pidf:'. It MAY contain other namespace declarations for the extensions used in the presence XML document.
2. <device> (RFC5491 [57])
   The <device> element […] can appear as a child to <presence>. There can be zero or more occurrences of this element per document. Each <device> element has a mandatory "id" attribute, which contains the occurrence identifier for the device. In the TR-069 framework the id attribute will contain the CWMP Identifier of the device, in the form OUI-ProductClass-SerialNumber.
3. <geopriv> (RFC5491 [57], RFC5139 [58])
   Location information in a PIDF-LO can be described in a geospatial manner based on a subset of Geography Markup Language (GML) 3.1.1 or as civic location information specified in RFC5139 [58]. The PIDF-LO Geodetic Shapes specification provides a specific GML profile for expressing commonly used shapes using simple GML representations. This profile defines eight shape types, the simplest ones being a 2-D and a 3-D Point. The PIDF-LO Geodetic Shapes specification also mandates the use of the World Geodetic System 1984 (WGS84) coordinate reference system and the usage of European Petroleum Survey Group (EPSG) code 4326 (as identified by the URN urn:ogc:def:crs:EPSG::4326) for two-dimensional (2d) shape representations and EPSG 4979 (as identified by the URN urn:ogc:def:crs:EPSG::4979) for three-dimensional (3d) volume representations.
   Each <geopriv> element must contain at least the following two child elements: <location-info> element and <usage-rules> element. One or more elements containing location information are contained inside a <location-info> element.
   a. <location-info> element can contain one or more elements bearing location information.

i. <Point> element contains geographical data in the coordinate system specified by its srsName attribute. In the example above (WGS84/EPSG 4326), the syntax is latitude, longitude expressed in degrees
ii. Civic information elements are specified by IETF and can be added to the geographical data, though mixing information is not recommended.
iii. <relative-location> element is being proposed by IETF

b. <usage-rules> can contain the following optional elements:
i. <retransmission-allowed>: When the value of this element is 'no', the recipient of this Location Object is not permitted to share the enclosed Location Information, or the object as a whole, with other parties. RFC4119 [56] specifies that "by default, the value MUST be assumed to be 'no'".
ii. <retention expires>: This field specifies an absolute date at which time the Recipient is no longer permitted to possess the location information
iii. <external ruleset>: This field contains a URI that indicates where a fuller ruleset of policies, related to this object, can be found
iv. <notewell>: This field contains a block of text containing further generic privacy directives.

c. <method> is an optional element that describes the way that the location information was derived or discovered. Values allowed by RFC4119 [56] are stored in the IANA registry as "Method Tokens" [60]. The "Wiremap" value listed in the example is described as "Location determined using wiremap correlations to circuit identifiers "

4. <deviceID> element is mandatory. It contains a globally unique identifier, in the form of a URN, for each of the presentity devices (RFC4479 [59])
5. <timestamp> is optional (RFC4479 [59])

### XVIII.3.3 Use of RFC5491 and RFC5139 Location XML Elements in CWMP or USP

1. <presence>
The entity attribute conveys no useful information and its value should be conventionally set to an empty string.
2. <device>
In RFC5491 [57] this is one of the devices associated to the presentity. Devices are identified in the presence document by means of an instance identifier specified in the id attribute.
3. <geopriv>
    a. <location-info>
    2-D geographical coordinates with no additional civic information are sufficient in the simplest case.
        - <Point>
        For 2-D applications the value of the srsName attribute should be set to the specified value "urn:ogc:def:crs:EPSG::4326"
    b. <usage-rules>

- <retransmission-allowed>
  Note that this field is not intended as instruction to the device whose location this is. Rather, it is intended to provide instruction to other systems that the device sends its location to (via SIP or other mechanisms). Therefore, the device will need to maintain its own policy (no standardized TR-069 data model is provided for this) as to when and where to send its location to others, and how to set this parameter when transmitting this location information. The device can choose to set this parameter to "yes" or to "no" when sending its location to others. RFC4119 [56] specifies that this element's default value is "no".

c. <method> If this location object is being created by the device as a result of GPS, A-GPS, or Manual mechanisms, the <method> parameter will be populated with "GPS", "A-GPS", or "Manual", respectively. If the location object is being created by External:CWMP, then this parameter will not be used or populated by the Controller.

4. <deviceID> It contains a globally unique identifier, in the form of a URN, for each of the presentity devices (RFC4479 [59]).
5. <timestamp> is optional. The device (GPS, A-GPS, Manual), ACS (External:CWMP) or USP-Controller (External:USP) can set this to the time the location was set or acquired.

# Appendix XIX Fault Management

This section discusses the Theory of Operation for Fault Management using CWMP [2] or USP [67] and the FaultMgmt object defined in the Root data model.

## XIX.1 Overview

There are four types of alarm event handling:

| | |
|---|---|
| Expedited Event | Alarm event is immediately notified to the Controller with the use of Active Notification mechanism |
| Queued Event | Alarm event is notified to the Controller at the next opportunity with the use of Passive Notification mechanism |
| Logged Event | The CPE stores the alarm event locally but does not notify the Controller |
| Disabled Event | The CPE ignores the alarm event and takes no action |

Note that all Fault Management tables are cleared when the device reboots.

Table 14 shows the multi-instance objects for FM to manage the alarm events.

**Table 14 – FM Object Definition**

| Object name (<rootobject>.*Fault Mgmt.*) | Table size | Content | Purpose and usage |
|---|---|---|---|
| SupportedAlarm.{i}. | Fixed | Static & fixed content | Defines all alarms that the CPE supports. *ReportedMechanism* defines how the alarm is to be handled within the CPE: *0 – Expedited, 1 – Queued, 2 – Logged, 3 – Disabled*<br>The table size is fixed and its content is static in order to drive the alarm handling behavior in the CPE. |
| ExpeditedEvent.{i}. | Fixed | Dynamically updated | Contains all *"Expedited"* type alarm events since the last device initialization. This includes events that are already reported or not yet reported to the Controller. One entry exists for each event. In other words, raising and clearing of the same alarm are two separate entries. As the table size is fixed (vendor defined), new alarm event overwrites the oldest entry in FIFO fashion after the table becomes full. |
| QueuedEvent.{i}. | Fixed | Dynamically updated | Contains all *"Queued"* type alarm events since the last device initialization. This |

| Object name (<rootobject>.*Fault Mgmt.*) | Table size | Content | Purpose and usage |
|---|---|---|---|
| | | | includes events that are already reported or not yet reported to the Controller. One entry exists for each event. In other words, raising and clearing of the same alarm are two separate entries.<br>As the table size is fixed (vendor defined), new alarm event overwrites the oldest entry in FIFO fashion after the table becomes full. |
| CurrentAlarm.{i}. | Variable | Dynamically updated | Contains all the currently active alarms (i.e. outstanding alarms that are not yet cleared) since the last device initialization. When an outstanding alarm is cleared, that entry is deleted from this table. Therefore, only 1 entry exists for a given unique alarm.<br><br>A Controller can retrieve the content of this table to get the entire view of the currently outstanding alarms.<br><br>As this is a variable size table, the size changes as alarm event is raised and cleared.<br><br>If maximum entries for this table are reached, the next event overrides the object with instance number 1. Subsequent entries override objects at sequentially increasing instance numbers. This logic provides for automatic "rolling" of records.<br>When a new alarm replaces an existing alarm, then all parameter values for that instance are considered as changed for the purposes of value change notifications to the Controller (even if their new values are identical to those of the prior alarm). |
| HistoryEvent.{i}. | Fixed | Dynamically updated | Contains all alarm events as a historical record keeping purpose. One entry exists for each event. In other words, raising and clearing of the same alarm are two separate entries.<br>The Controller can retrieve the content of this table to get the entire chronological history of the alarm events on the CPE.<br>As the table size is fixed (vendor defined), |

| Object name (<rootobject>.*Fault Mgmt.*) | Table size | Content | Purpose and usage |
|---|---|---|---|
| | | | new alarm event overwrites the oldest entry in FIFO fashion after the table becomes full. |

# Appendix XX BASAPM and LMAP Theory of Operation

Broadband Access Service Attributes and Performance Metrics (BASAPM) and Large-Scale Measurement of Broadband Performance (LMAP) data model components are derived from TR-304 [63] and the IETF LMAP information model [65], respectively.

## XX.1 TR-069 Family of Specifications in the Context of TR-304

This section describes possible deployment scenarios where the CWMP and IPDR protocols are used for the respective TR-304 protocols.

### XX.1.1 TR-304 and IETF LMAP Frameworks

The IETF (LMAP) and BBF (TR-304) use a similar framework for diagnostics where each framework consists of a Measurement Controller, Data Collector and Measurement Agent. While there are differences between TR-304 and LMAP elements in various deployment scenarios, in residential scenarios the behavior of Measurement Agent in the home is consistent between the IETF LMAP and BBF TR-304 frameworks.

#### XX.1.1.1 TR-304 Framework

The TR-304 framework consists of a Management Server that is used to manage and configure the Measurement Agent. This would also include receiving logging and status information as well as the capability to schedule the Measurement Agent for tests. The TR-304 framework also has a Measurement Controller with the responsibility to schedule the Measurement Agent for tests to be performed provide test admin control. TR-304 framework also has multiple channels where a Measurement Agent can send reports to the different Data Collectors.

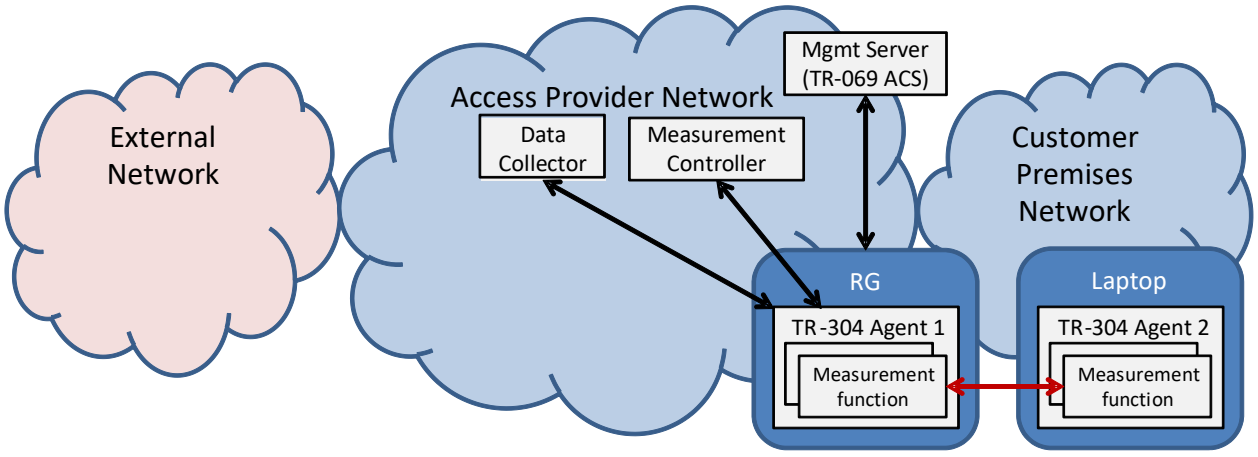



**Figure 66 – TR-304 Framework**

### XX.1.1.2 IETF LMAP Framework

The IETF LMAP framework, like the BBF TR-304 framework, consists of a Management Server that is used to pre-configure the Measurement Agent. Note that this also could be done at the manufacturing stage of the device. The LMAP framework also has a Measurement Controller with the responsibility to configure the Measurement Agent for tests to be performed; provide instructions about the test and receive status and logging information the Measurement agents. In the IETF LMAP framework these functions are treated as individual channels that can be assigned to different Measurement Controllers. Likewise the Reporting interface also has multiple channels where a Measurement Agent can send reports to the different Data Collectors.

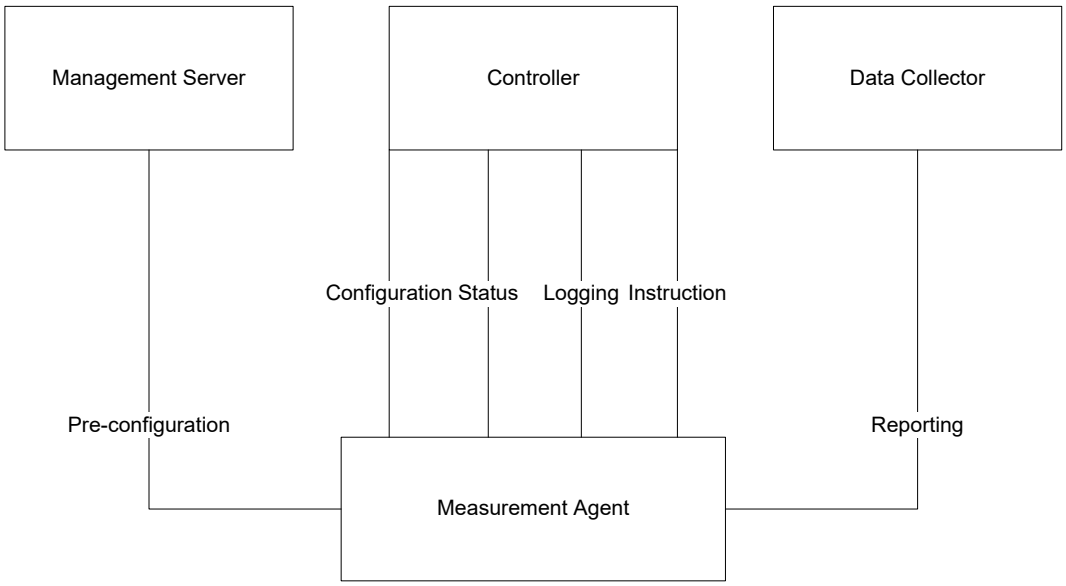



**Figure 67 – LMAP Framework**

### XX.1.2 CWMP for Pre-configuration

In the IETF LMAP and TR-304 frameworks, CWMP can be used to pre-configure the Measurement Agent; where the Controller and Data Collector could use other protocols (e.g., IETF LMAP protocol).

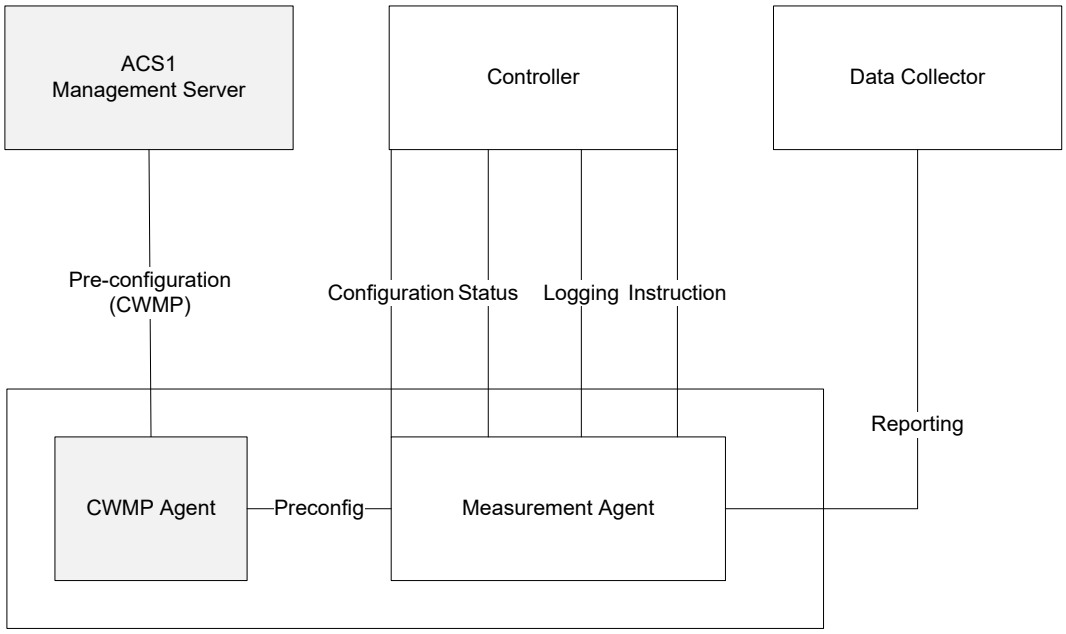



**Figure 68 – CWMP for Pre-configuration**

Note that in the TR-304 framework the Status and Logging functions have not been explicitly identified as capabilities of the Controller.

### XX.1.3 CWMP for Control and Pre-configuration, IPDR for Reporting

In the IETF LMAP and TR-304 frameworks, CWMP can be used to pre-configure the Measurement Agent and manage/schedule the tests. Likewise the IPDR protocol can be used to report the test results. In this scenario, the ACS would act as the Management Server and Measurement Controller. This scenario would place a constraint on the IETF LMAP framework in that there would be allowed only 1 Measurement Controller per Measurement Agent. See Section XX.3 for additional information on use of the BulkData.Profile object in the context of LMAP.

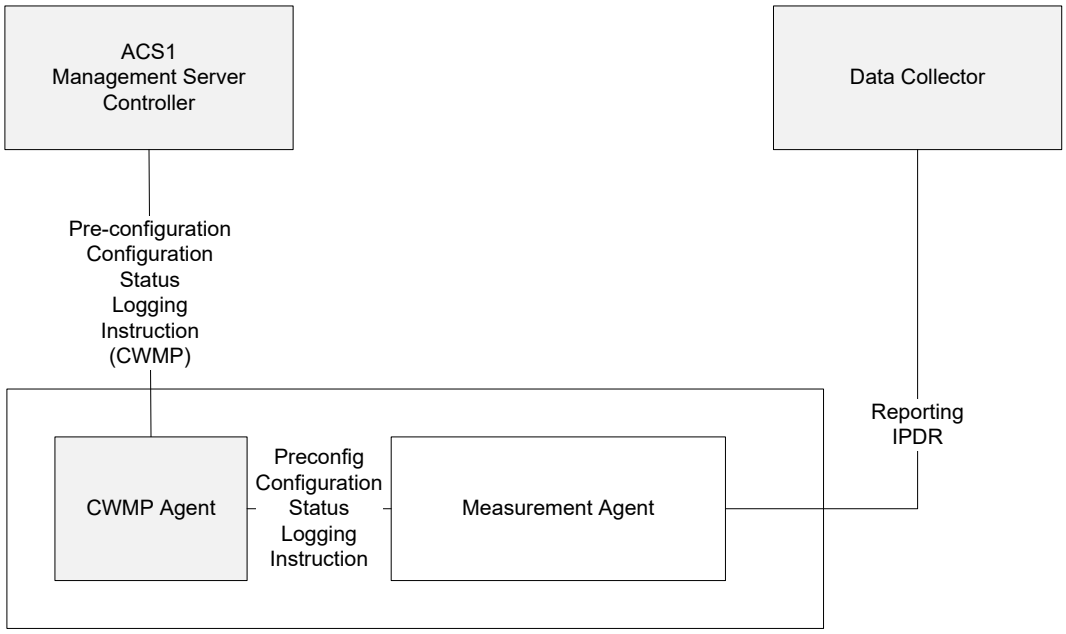



**Figure 69 – CWMP for Control and Pre-configuration, IPDR for Reporting**

### XX.1.4 CWMP as a Proxier, IPDR for Reporting

In scenarios where Measurement Agent does not have connectivity with the Measurement Controller, CWMP can be used to act as a proxy between the Measurement Controller and Measurement Agent. In this scenario, if the CWMP Proxy is an Embedded Device then both Measurement Agents are associated with the same ACS. If the Measurement Agents need to be associated with different Measurement Controllers then the CWMP Virtual Device mechanism is to be used.

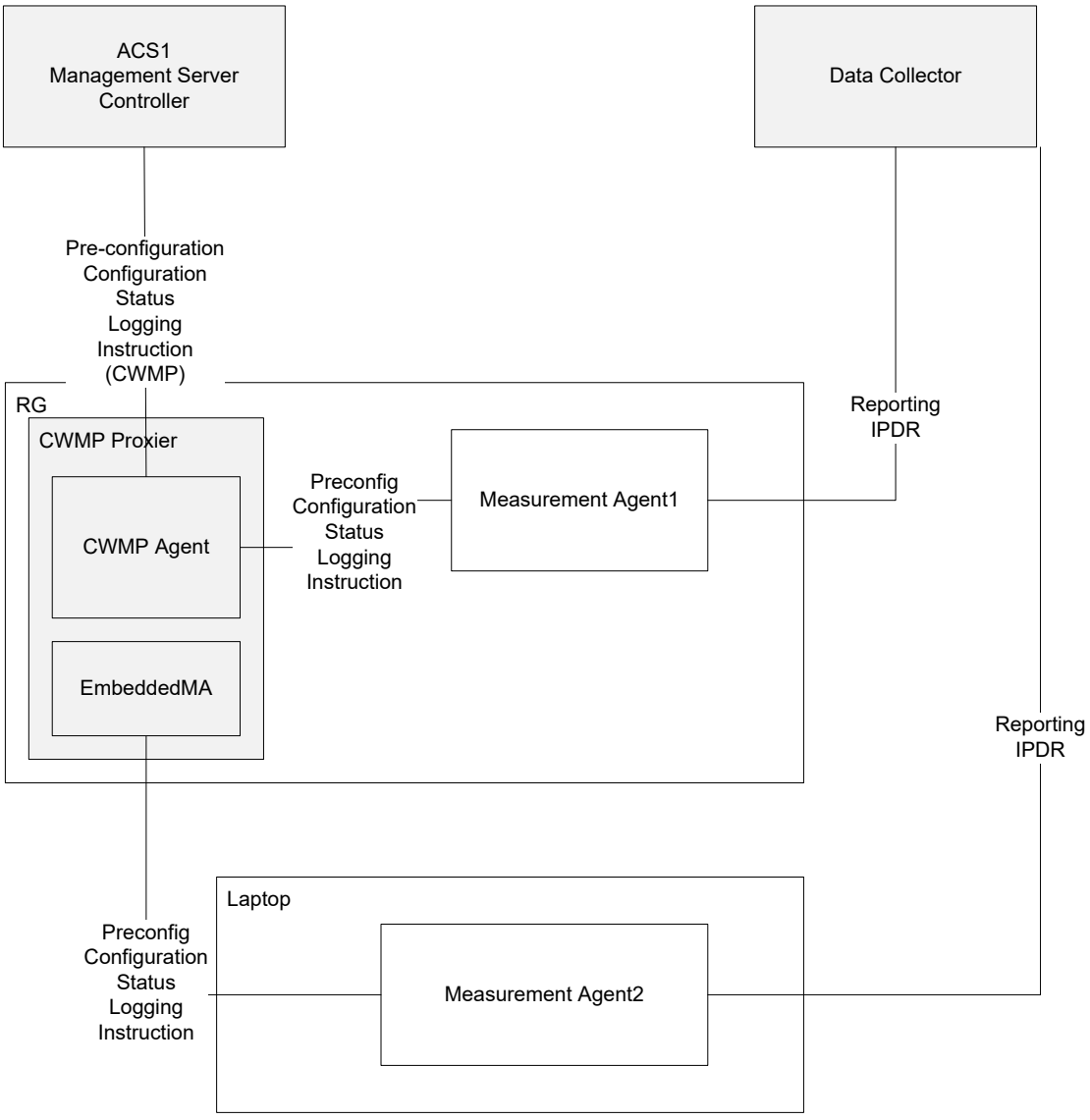



**Figure 70 – CWMP Proxy Device Deployment**

## XX.1.5 Multi-ACS Deployment

In the IETF LMAP framework, the Measurement Agent could interact with different elements that implement the functionality of the Management Server and Measurement Controller. In addition, the IETF LMAP framework also permits the functionality of the Measurement Controller to be implemented in multiple elements.
For a CWMP framework, this would require a different CWMP Agent for each application. As such this type of scenario is not realistically supported by CWMP.

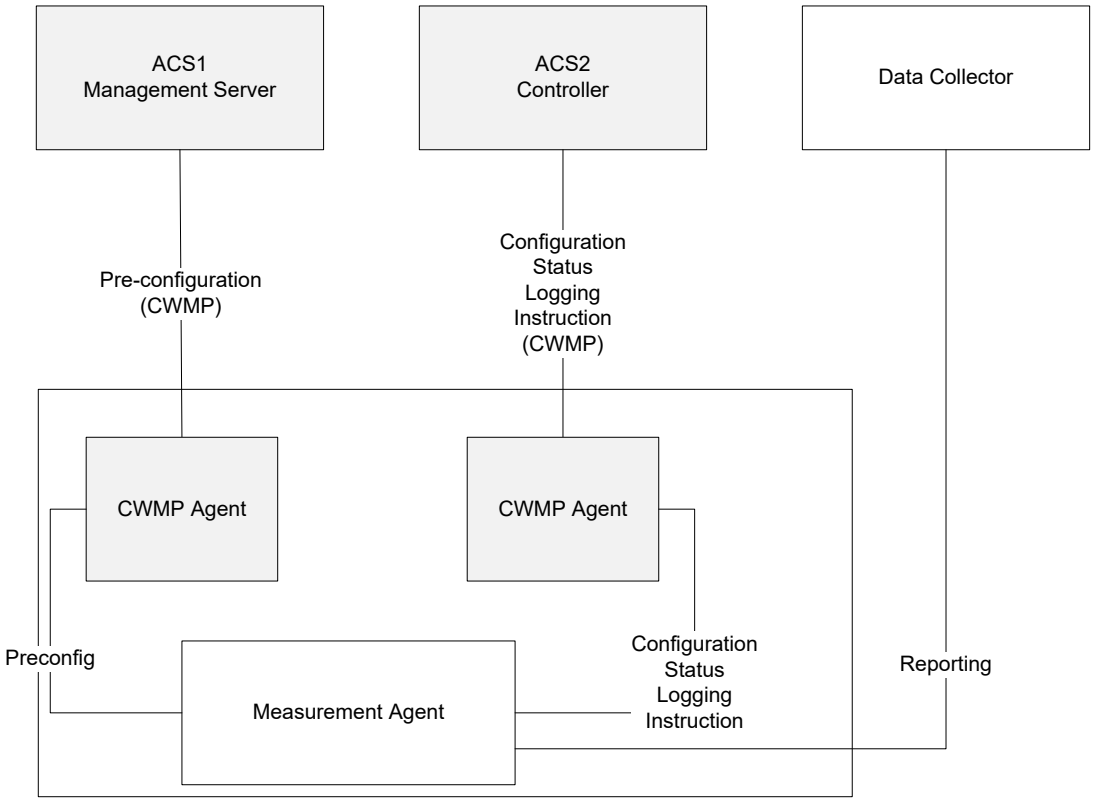



**Figure 71 – CWMP Multi-ACS Deployment**

## XX.2 Derivation of Data Model Elements

### XX.2.1 Device.BASAPM

Device.BASAPM provides a TR-304 [63] wrapper for a Device.LMAP. MeasurementAgent instance. Device.BASAPM provides parameters related to the operational domain, device ownership, device identification, geographic location, and measurement reference point of a referenced Device.LMAP.MeasurementAgent instance.

### XX.2.2 Device.LMAP.MeasurementAgent

The Device.LMAP objects and parameters are mostly described in the IETF LMAP information model [65]. That document serves as the primary vehicle for describing theory of operations for Device.LMAP.MeasurementAgent.

The base Device.LMAP.MeasurementAgent.{i} object contains parameters defined in LMAP information model ma-config-obj, ma-status-obj, and ma-capability-obj. The ma-preconfig-obj parameters are not modeled in Device:2 data model , because there is no need for pre-configuration values in a CWMP/USP-managed Measurement Agent. The information model parameters map to Device:2 data model parameters as shown in Table 15:

**Table 15 – Mapping LMAP Information Model Parameters to Data Model Parameters**

| IETF LMAP Information Model Parameter | Device:2 data model parameter (in Device.LMAP.MeasurementAgent.{i}) |
|---|---|
| ma-config-agent-id | Identifier |
| ma-config-credentials | PublicCredential, PrivateCredential |
| ma-config-group-id | GroupIdentifier |
| ma-config-measurement-point | MeasurementPoint |
| ma-config-report-agent-id | UseAgentIdentifierInReports |
| ma-config-report-group-id | UseGroupIdentifierInReports |
| ma-config-report-measurement-point | UseMeasurementPointInReports |
| ma-config-controller-timeout | Controller. ControllerTimeout |
| ma-status-last-started | LastStarted |
| ma-capability-hardware | not included in Device.LMAP because it duplicates Device.DeviceInfo.HardwareVersion |
| ma-capability-firmware | not included in Device.LMAP because it duplicates Device.DeviceInfo.SoftwareVersion |
| ma-capability-version | Version |
| ma-capability-tags | CapabilityTags |

All of the other IETF LMAP information model parameters can be readily mapped to objects and parameters in Device.LMAP.MeasurementAgent.{i}.

## XX.3 Bulk Data Collection in the Context of LMAP

The TR-069 family of specifications has defined protocols that can be used for the collection of bulk data between a CWMP Agent and an ACS. These protocols are defined for IPDR [62] and HTTP [2]. The Device:2 data model described in Section XX.2 includes the ability to use these protocols for transferring test results between a Measurement Agent and a Data Collector.

When integrating the test results of the Device:2 data model (i.e., LMAP.Report object instance) into the bulk data objects and parameters provided by the Device:2 data model, the LMAP.Report object instance becomes the referenced parameter of the Bulk Data Profile (BulkData.Profile object instance). In addition, there is a linkage needed within the LMAP data model to identify the BulkData.Profile object instance. This is done through the reference of the BulkData.Profile object instance from the LMAP data model's Communication Channel for a Scheduled Action.

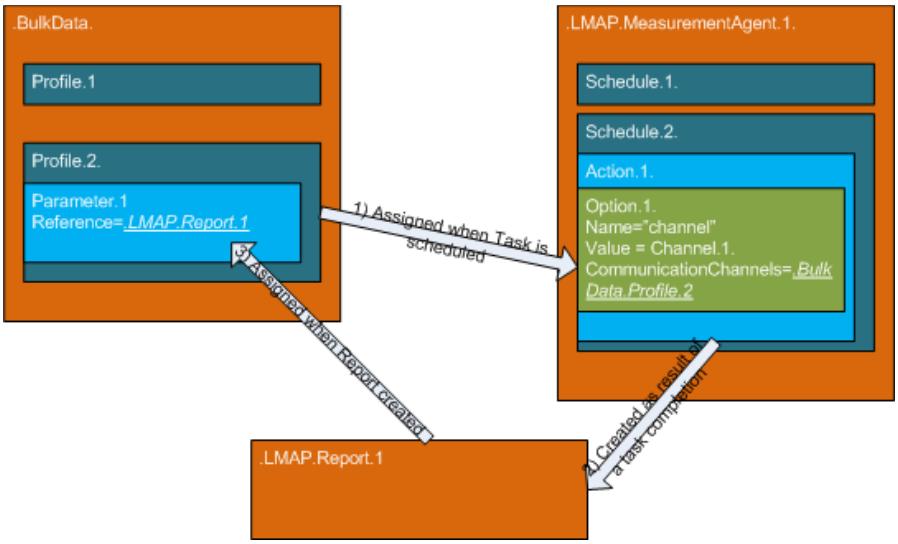



**Figure 72 – Integration of Bulk Data Profiles with LMAP**

## XX.4 TR-143 Diagnostics in LMAP

TR-143 [61] describes a set of tests that can be used within the context of TR-304 based on the IETF LMAP framework [64] and Information Model [65] and implemented using the Device:2 data model in section XX.2. These tests could be defined using the following procedure:

1. The TR-143 diagnostic needs to be identified as a URI in the registry entry (Device.LMAP.MeasurementAgent.{i}.TaskCapability.{i}.Registry.{i}.RegistryEntry):
   - The URI is in the form of: urn:bbf:lmap:<BBF TR>:<DiagnosticProfileName>
   - For example a TR-143 upload diagnostic could be: “urn:bbf:lmap:tr-181-2-11-0:UploadDiagnostics-1”
2. The TR-143 diagnostic’s parameters and objects that are modifiable by the Controller/Measurement Controller are encoded in the Device.LMAP.MeasurementAgent.{i}.Task.{i}.Option.{i}. or Device.LMAP.MeasurementAgent.{i}.Schedule.{i}.Action.{i}.Option.{i} objects.
   - For example: Device.IP.Diagnostics.UploadDiagnostics.DiagnosticsState=requested
3. The TR-143 diagnostic’s parameters and objects that are read-only are encoded in the Device.LMAP.Report.{i}.Task.{i}.Result.{i}.Values where each parameter name is encoded in the Device.LMAP.Report.{i}.Task.{i}.ColumnLabels parameter.
   - For example: ColumnLabels: Device.IP.Diagnostics.UploadDiagnostics.PerConnectionResult.{1}.TotalBytesSent
   - Value: 30

*NOTE – These fully qualified names could be shortened or even specified as a different name based on the specification behind the RegistryEntry URN.*

End of Broadband Forum Technical Report TR-181