UPnP-QoS Architecture:3

For UPnP Version 1.0
Status: Standardized DCP
Date: November 30, 2008

Authors	Member
Ally Yu-kyoung Song	LGE
Amol Bhagwat	CableLabs
Bruce Fairman	Sony
Daryl Hlasny	Sharp Labs of America
Dieter Verslype	Ghent University
Fred Tuck (co-chair)	EchoStar
Jelle Nelis	Ghent University
Michael van Hartskamp (co-chair)	Philips
Narm Gadiraju	Intel Corporation
Puneet Sharma	HP
Richard Chen	Philips
Sherman Gavette	Sharp Labs of America
Steve Wade	Sharp Labs of America
Suman Sharma	Intel Corporation

Authors	Member
Zong Wu	Entropic

The UPnP Forum in no way guarantees the accuracy or completeness of this author list and in no way implies any rights for or support from those members listed. This list is not the specifications' contributor list that is kept on the UPnP Forum's website.

Contents

1. INTRODUCTION 6

1.1. VERSIONS OF THE UPNP-QOS SPECIFICATIONS 6
1.2. INFORMATIVE REFERENCES 6

2. ARCHITECTURE OVERVIEW 9

2.1. SCOPE 9
2.2. ASSUMPTIONS 9
2.3. ARCHITECTURE SUMMARY 9

3. ARCHITECTURE 11

3.1. POLICY-BASED QOS 11
3.2. TYPES OF QOS 11
3.2.1. Prioritized QoS 11
3.2.2. Parameterized QoS 12
3.2.3. Hybrid QoS 14

4. KEY CONCEPTS AND EXAMPLES 15

4.1. INTERFACES AND LINKS 15
4.2. PATH INFORMATION 15
4.2.1. Example – Bridge on the path 16
4.2.2. Example – Device not on the path 16
4.2.3. Example – Path Determination 16
4.3. QOS SEGMENTS 17
4.3.1. Example – Simple QoS Segment 18
4.3.2. Example – Multiple QoS Segments 18
4.3.3. Example – Homogeneous QoS Segment with L2 QoS Bridges 19
4.3.4. Example – Heterogeneous QoS Segment with L2 QoS Bridges 20
4.3.5. QosSegmentId generation examples 20
4.4. ADJACENCY OF QOSDEVICE SERVICES 21

5. UPNP-QOS SERVICES 23

5.1. THE QOSPOLICYHOLDER SERVICE 23
5.1.1. Overview 23
5.1.2. Traffic Stream QoS Policy Description 23
5.1.3. Multiple instances of the QosPolicyHolder Services 24
5.1.4. Preferred QosPolicyHolder Service 24
5.1.5. Maintaining the Preference of a QosPolicyHolder Service 24
5.1.6. Configuring the QosPolicyHolder Service 25
5.2. THE QOSMANAGER SERVICE 25
5.2.1. Overview 25
5.2.2. Behavior 26
5.2.3. Update the QoS reservation 26
5.3. THE QOSDEVICE SERVICE 27
5.3.1. Overview 27
5.3.2. Behavior 27
5.3.3. Configuring QoS 28
5.3.4. Path Information 28
5.3.5. Ancillary actions 28
5.3.6. Events 29

6. SYSTEM OPERATION 30

6.1. SELECTION OF A QOSMANAGER SERVICE 30
6.2. INVOKING THE QOSMANAGER SERVICE 31
6.2.1. Initiation of QoS Setup (I) 31
6.2.2. Initiation QoS Setup (II) 31
6.2.3. Release of QoS Resources 32
6.2.4. Changing the QoS Setup 32
6.2.5. Integrated Control Point 32
6.2.6. Independent AV Control Point 33
6.2.7. Determination of QosBoundary Source and Destination 33
6.2.8. Creation of the TSPEC (Traffic Specification) 34
6.3. DETERMINATION OF POLICY FOR THE TRAFFIC STREAM 34
6.3.1. Preferred QosPolicyHolder Service 34
6.3.2. CP-Indicated QosPolicyHolder Service 34
6.3.3. Single QosPolicyHolder Service 34
6.3.4. Priority Order of QosPolicyHolder Services for Prioritized QoS 34
6.3.5. Priority Order of QosPolicyHolder Services for Parameterized QoS and Hybrid QoS 35
6.3.6. The QosPolicyHolder Service 35
6.3.7. Default Policy 35
6.4. DETERMINATION OF QOSDEVICE SERVICES THAT HAVE TO BE MANAGED 36
6.4.1. Configuration of QoS Devices 36
6.4.2. Path Determination 36
6.4.3. QoS Segment Identification 36
6.5. ADMISSION CONTROL 37
6.5.1. Decomposition of End-to-End Requirements into Per-QoS Segment Requirements 37
6.5.2. Determination of adjacent QosDevice services within a QoS Segment 38
6.5.3. Configuring QosDevice Services within a QoS Segment – release 39
6.5.4. Configuring QosDevice Services within a QoS Segment 39
6.5.5. Device resources managed by the QosDevice Service 40
6.5.6. Collecting the results of all QoS Segments 40
6.5.7. The QosDevice Service and the QD:AdmitTrafficQos() action 41
6.5.8. The QosDevice Service and the QD:ReleaseAdmittedQos() action 42
6.5.9. The QosDevice Service and the QD:UpdateAdmittedQos() action 42
6.6. PREEMPTION 43
6.6.1. Identifying the Blocking Traffic Streams 43
6.6.2. Determining Candidates for Preemption 43
6.6.3. The Preemption and notification 45
6.6.4. Re-Attempt To Admit the Traffic Stream 45
6.7. RUN TIME OPERATION 45
6.7.1. Traffic Lease Management and Link failures 45
6.7.2. Violation and Policing of the TSPEC 46
6.7.3. Being Preempted 46

7. QOS BOUNDARY ADDRESSES 47

List of Figures

Figure 1: UPnP-QoS Architecture Overview....10
Figure 2: An Example Interaction Diagram for *RequestTrafficQos()* action for prioritized QoS setup.....12
Figure 3: An Example Interaction Diagram for RequestTrafficQos() (without preemption).....13
Figure 4: An Example Interaction Diagram for RequestExtendedTrafficQos() with preemption capability. 13
Figure 5: Example of Interfaces and Links. Device A has 1 interface that contains 3 links. Device B, C, and D each contain an interface with only a single link.....15
Figure 6: A bridge is on the path if and only if it reports the MAC address of the source on a different link than the MAC address of the sink and these two links are bridged.....16
Figure 7: Laptop is not bridging its interfaces and therefore not on the path.....16
Figure 8: Example network for path determination 17
Figure 9: A simple network with one QoS Segment 18
Figure 10: A network with two different technologies and two QoS Segments.....19
Figure 11: A network with Ethernet Layer-2-QoS bridges, L2Q-end point devices and legacy Ethernet bridges and legacy Ethernet devices 19
Figure 12: A network with all L2Q-end point devices but with different underlying technologies: MoCA (left hand side) and Ethernet (Right hand side), respectively 20
Figure 13: QoS Segment with two *QosDevice* services.....21
Figure 14: QoS Segment with only one *QosDevice* Service.....22
Figure 15: Example of Adjacent *QosDevice* services.....38
Figure 16: Example of Adjacent *QosDevice* Services. Note that only A and C are *QosDevice* Services.....39
Figure 17: Examples of approaches to determine candidates for preemption.....44

1. Introduction

This architecture document describes the motivation, use and interaction of the three services that comprise version 3 of the UPnP-QoS Framework:

- The *QosDevice:3* Service [QD:3],
- The *QosPolicyHolder:3* Service [QPH:3], and
- The *QosManager:3* Service [QM:3].

While UPnP-QoS defines three services (listed above), it does not define a new device type. Since Quality of Service issues need to be solved for multiple usage scenarios, it is expected that vendors could use any UPnP device as a container for the services defined by UPnP-QoS.

The UPnP-QoS Framework is compliant with the UPnP Device Architecture version 1.0.

This document is INFORMATIVE. This document is derived from the specifications and it does not describe what is required or optional. For required and optional functionalities, refer to the service definition documents. When there is a conflict between this document and (one of) the service definition documents, the latters prevail. This document avoids the words must, should, and may. Implementers are therefore referred to the appropriate service definition documents for requirements.

Definitions for terms used in this document can be found in [QM:3].

1.1. Versions of the UPnP-QoS Specifications

There are currently three versions of UPnP-QoS.

UPnP-QoS version 1 defines a framework for policy-based prioritized QoS.

UPnP-QoS version 2 extends the version 1 framework with

- A rotameter service to measure network performance and assist in the diagnosis of network problems, and
- A mechanism to indicate a *QosPolicyHolder* Service that will be used by the *QosManager* Service

UPnP-QoS version 3 extends the version 2 framework with

- Support for Admission Control,
- A mechanism to support a Preferred *QosPolicyHolder* Service,
- A way to configure the policy in the *QosPolicyHolder* Service.

This document describes UPnP-QoS version 3, please refer to the version 2 [QA:2] or version 1 [QA:1] architecture documents for more information on those versions.

1.2. Informative References

[Annex_G] –IEEE Std 802.1Q, Annex G, *IEEE Standard for Information technology - Telecommunications and information exchange between systems - IEEE standard for local and metropolitan area networks - Common specifications - Media access control (MAC) Bridges*, 2018.

[XML] – *Extensible Markup Language (XML) 1.0 (Second Edition)*, T. Bray, J.Paoli, C. M. Sperberg-McQueen, E Maler, eds. W3C Recommendations, 6 October 2000.
Available at: http://www.w3.org/TR/2000/REC-xml-20001006
Latest version available at: http://www.w3.org/TR/REC-xml/

[QM:1] – *UPnP QosManager:1* Service Document,
Available at: http://www.upnp.org/standardizeddcps/documents/UPnP_Qos_Manager1_000.pdf

[QM:2] – *UPnP QosManager:2* Service Document,
Available at: http://www.upnp.org/specs/qos/UPnP-qos-QosManager-v2-Service-20061016.pdf
Latest version available at: http://www.upnp.org/specs/qos/UPnP-qos-QosManager-v2-Service.pdf

[QM:3] – *UPnP QosManager:3* Service Document,
Available at: http://www.upnp.org/specs/qos/UPnP-qos-QosManager-v3-Service-20081130.pdf
Latest version available at: http://www.upnp.org/specs/qos/UPnP-qos-QosManager-v3-Service.pdf

[QD:1] – *UPnP QosDevice:1* Service Document
Available at: http://www.upnp.org/standardizeddcps/documents/UPnP_Qos_Device1_000.pdf

[QD:2] – *UPnP QosDevice:2* Service Document
Available at: http://www.upnp.org/specs/qos/UPnP-qos-QosDevice-v2-Service-20061016.pdf
Latest version available at: http://www.upnp.org/specs/qos/UPnP-qos-QosDevice-v2-Service.pdf

[QD:3] – *UPnP QosDevice:3* Service Document
Available at: http://www.upnp.org/specs/qos/UPnP-qos-QosDevice-v3-Service-20081130.pdf
Latest version available at: http://www.upnp.org/specs/qos/UPnP-qos-QosDevice-v3-Service.pdf

[QDA:3] – *UPnP QosDevice:3* Underlying Technology Interface Addendum
Available at: http://www.upnp.org/specs/qos/UPnP-qos-QosDevice-v3-Addendum-20081130.pdf
Latest version available at: http://www.upnp.org/specs/qos/UPnP-qos-QosDevice-v3-Addendum.pdf

[QPH:1] – *UPnP QosPolicyHolder:1* Service Document
Available at: http://www.upnp.org/standardizeddcps/documents/UPnP_Qos_Policy_Holder1.pdf

[QPH:2] – *UPnP QosPolicyHolder:2* Service Document
Available at: http://www.upnp.org/specs/qos/UPnP-qos-QosPolicyHolder-v2-Service-20061016.pdf
Latest version available at: http://www.upnp.org/specs/qos/UPnP-qos-QosPolicyHolder-v2-Service.pdf

[QPH:3] – *UPnP QosPolicyHolder:3* Service Document
Available at: http://www.upnp.org/specs/qos/UPnP-qos-QosPolicyHolder-v3-Service-20081130.pdf
Latest version available at: http://www.upnp.org/specs/qos/UPnP-qos-QosPolicyHolder-v3-Service.pdf

[AV]– UPnP AV Architecture:1 Document version 1.0
Available at: http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v1-20020625.pdf.

[DEVICE] – UPnP Device Architecture, version 1.0.
Available at: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20060720.pdf
Latest version available at: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf

[DSCP] – IETF RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, K. Nichols et al., December 1998.
Available at: http://www.ietf.org/rfc/rfc2474.txt

[RFC3339] – *Date and Time on the Internet: Timestamps*, G. Klyne et al., July 2002.
Available at: http://www.ietf.org/rfc/rfc3339.txt

[RFC3927] – *Dynamic Configuration of IPv4 Link-Local Addresses*. S. Cheshire et al., May 2005.
Available at: http://www.ietf.org/rfc/rfc3927.txt

[MoCA 1.0] – MoCA Mac/Phy Specification v1.0 2006.

[MoCA 1.1] – MoCA Mac/Phy Specification v1.1 Extensions 2007.

[HPAV] – HPAV HomePlug AV Specification Version 1.1.00.

[CDS:1] – UPnP AV ContentDirectory Service Definition document version 1.0.
Available at: http://www.upnp.org/standardizeddcps/documents/ContentDirectory1.0.pdf
Latest version available at:

[CDS:2] – UPnP AV ContentDirectory Service Definition document version 2.0.
Available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v2-Service.pdf
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v2-Service-20060531.pdf

[QA:1] – UPnP-QoS Architecture version 1.0
Available at: http://www.upnp.org/standardizeddcps/documents/UPnP_QoS_Architecture1.pdf

[QA:2] – UPnP-QoS Architecture version 2.0
Available at: http://www.upnp.org/specs/qos/UPnP-qos-Architecture-v2-20061016.pdf
Latest version available at: http://www.upnp.org/specs/qos/UPnP-qos-Architecture-v2.pdf

2. Architecture Overview

This section provides an overview of UPnP-QoS v3.

2.1. Scope

The UPnP-QoS specifications are designed with a network in mind that consists of a single IP subnet. Therefore routing between devices in the network is out of scope for UPnP-QoS.

The UPnP-QoS specifications support QoS setup for all traffic streams and UPnP-AV traffic streams in particular. The UPnP-QoS specifications also support QoS management on the LAN for traffic streams originating from or terminating in a WAN.

2.2. Assumptions

UPnP-QoS starts from the basic assumption that a typical UPnP network is heterogeneous in its Admission Mechanisms. Obviously, a homogeneous Admission Mechanism is also accommodated.

2.3. Architecture Summary

This section is a brief overview of the UPnP-QoS Architecture. In Chapter 3 more details are provided on the high-level architecture. Chapter 4 describes and illustrates the key concepts. Chapter 5 presents the three services in detail and Chapter 6 provides details on the interaction between Control Points and the services.

UPnP-QoS defines three services. These are the *QosPolicyHolder* Service [QPH:3], the *QosManager* Service [QM:3] and the *QosDevice* Service [QD:3]. All three services are shown in Figure 1. In this figure, the numbers in parenthesis indicate the order in which the UPnP-QoS actions are invoked.

To illustrate the relationships of various UPnP-QoS components, an example scenario with a simple sequence of setup steps is described below. The detailed operations are described in Chapter 6.

Fundamentally, UPnP-QoS manages the QoS for a traffic stream that flows between a source and a sink device. A traffic stream is viewed as a uni-directional flow from a source device to a sink device, possibly passing through intermediate devices.

In the interaction described in this section, a Control Point application is assumed to have the knowledge of source, sink and content characteristics to be streamed, along with the content's Traffic Specification (TSPEC). The Control Point constructs a *TrafficDescriptor* structure and requests a *QosManager* Service on the UPnP network to setup QoS for a traffic stream (step 1). The Control Point in the *QosManager* Service (from hereon referred to as QoS Manager) requests the *QosPolicyHolder* Service (step 2) to provide appropriate policy for the traffic stream described by the *TrafficDescriptor* structure (step 3). Based on this policy, the QoS Manager configures the *QosDevice* Service(s) for establishing the QoS for the new traffic stream (step 4).

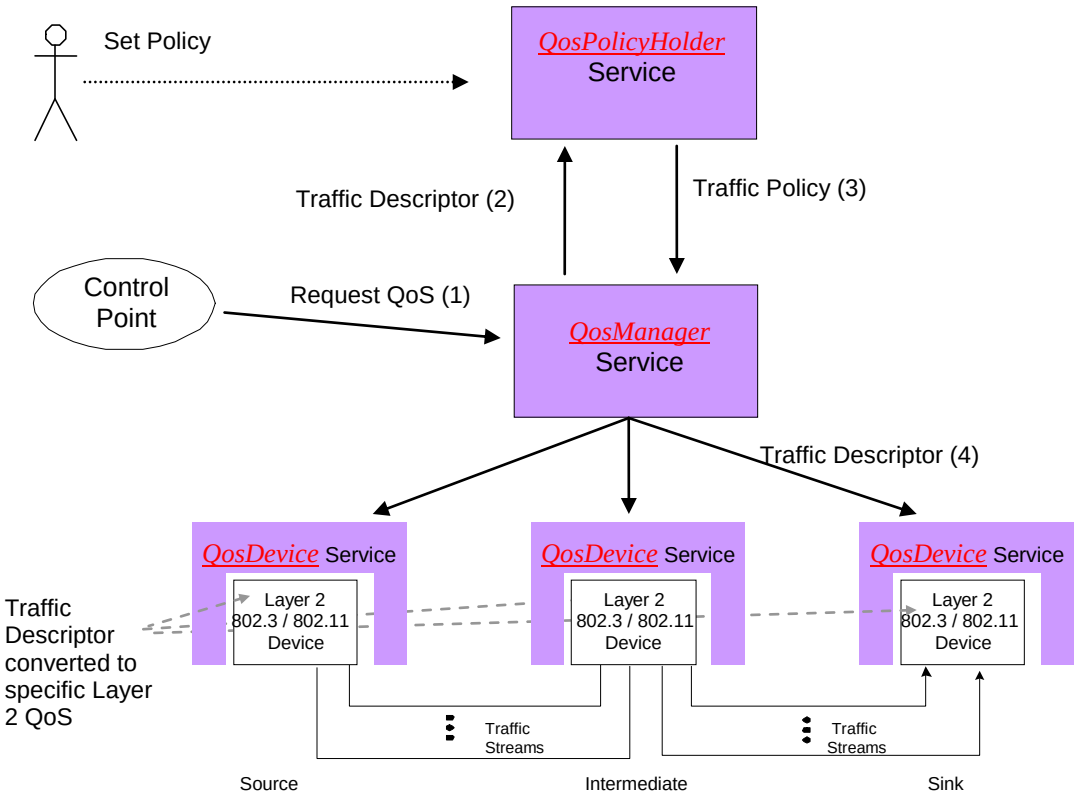

Figure 1: UPnP-QoS Architecture Overview

The *QosPolicyHolder* Service provides traffic policy for the UPnP network on which it resides. It is used to set the importance of a traffic stream by returning a *TrafficImportanceNumber* and a *UserImportanceNumber* (these are elements of the *TrafficPolicy* structure).

For requests for prioritized streams, the *TrafficDescriptor* structure containing the *TrafficImportanceNumber* is conveyed to *QosDevice* Service(s) by the QoS Manager and is used by the *QosDevice* Service to derive the technology specific L2 priority.

The internal mechanism used by the *QosDevice* Service for applying the *TrafficImportanceNumber* is L2 technology specific and is out of scope.

For requests to admit streams, the *TrafficDescriptor* structure containing the TSPEC is conveyed to the *QosDevice* Service(s) by the QoS Manager and is used by the *QosDevice* Service to request admission and possibly a reservation of resources in accordance with the requested TSPEC. If the admission fails because there are insufficient resources, the preferred *QosPolicyHolder* Service (see Section 5.1.4) can be consulted to determine the *UserImportanceNumber* for every Blocking UPnP stream. These numbers are used to rank Blocking traffic streams and to determine whether a new traffic stream can be admitted at the expense of existing Blocking traffic streams. The process of taking away resources from existing traffic streams to admit a new traffic streams based on their ranking is called preemption.

3. Architecture

3.1. Policy-based QoS

Policy-based QoS provides a way to assign priority for traffic streams and is also the basis for preemption.

A policy-based QoS system allows an individual or entity to define rules, based on traffic characteristics and to manage the traffic's QoS in the context of the policy system. These rules are then applied to the characteristics of a request to determine the QoS applied. The rules utilize characteristics such as network address or application type.

The *QosPolicyHolder* Service provides the mechanism for classifying and ranking traffic streams according to information provided with the action to request QoS for a particular traffic stream. The type of information provided in the *TrafficDescriptor* structure includes, among other items, traffic class (best-effort, video, voice, etc.), the source and destination IP addresses for the stream, the TSPEC structure, application name, username, etc. The *QosPolicyHolder* Service examines the information provided in the *TrafficDescriptor* structure and returns the importance, in the form of a *TrafficPolicy* structure.

3.2. Types of QoS

UPnP-QoS supports three types of QoS: Prioritized QoS, Parameterized QoS and Hybrid QoS. Prioritized QoS is the default. Prioritized QoS means end-to-end prioritized QoS. Parameterized QoS requests that network resources are reserved for a traffic stream end-to-end. If there is a non-parameterized QoS technology on the path, then parameterized QoS is not possible end-to-end on the entire path. Hybrid QoS addresses this issue. A request for Hybrid QoS is a request for parameterized QoS on QoS Segments that support parameterized QoS and prioritized QoS on other QoS Segments. If a Hybrid QoS admission fails on a parameterized QoS Segment then Hybrid QoS is not established and an explicit request for Prioritized QoS could be attempted by the Control Point. Note that a request for Hybrid QoS can also fail on a prioritized QoS Segment.

3.2.1. Prioritized QoS

This section describes the interaction between the services for a request for prioritized QoS. Figure 2 illustrates the interaction. First the Control Point composes a request to the *QosManager* Service based, for instance, on the information in the *ContentDirectory* Service [CDS:1]. Then the control point invokes the *QM:RequestTrafficQos()* action.

The QoS Manager collects information from the various *QosDevice* Services in the network. It obtains path information, with the *QD:GetPathInformation()* action or other QoS related information with the *QD:GetExtendedQosState()* action.

The *TrafficImportanceNumber* is determined by the *QosPolicyHolder* Service and returned following the *QPH:GetTrafficPolicy()* request. Based on the information available in the traffic descriptor (in particular Traffic Class) the *QosPolicyHolder* Service provides the *TrafficImportanceNumber*. In the case where the *QosPolicyHolder* Service cannot be used, *TrafficImportanceNumber* is determined by the QoS Manager using default policies.

The QoS Manager communicates with the *QD:SetupTrafficQos()* and *QD:AdmitTrafficQos()* actions to the *QosDevice* Services with a *TrafficDescriptor* structure containing a *TrafficImportanceNumber* consistent with the QoS Policy. The following diagram depicts the sequence of messaging for the setting up QoS for a given traffic stream.

The mapping of the *TrafficImportanceNumber* to the priority tag value used for the layer-2 network is dependent on the particular L2 technology. The *TrafficImportanceNumber* has been defined to be

consistent with the IEEE 802.1Q, Annex G, so it is expected to be a one-to-one mapping from the *TrafficImportanceNumber* to the priority tag value.

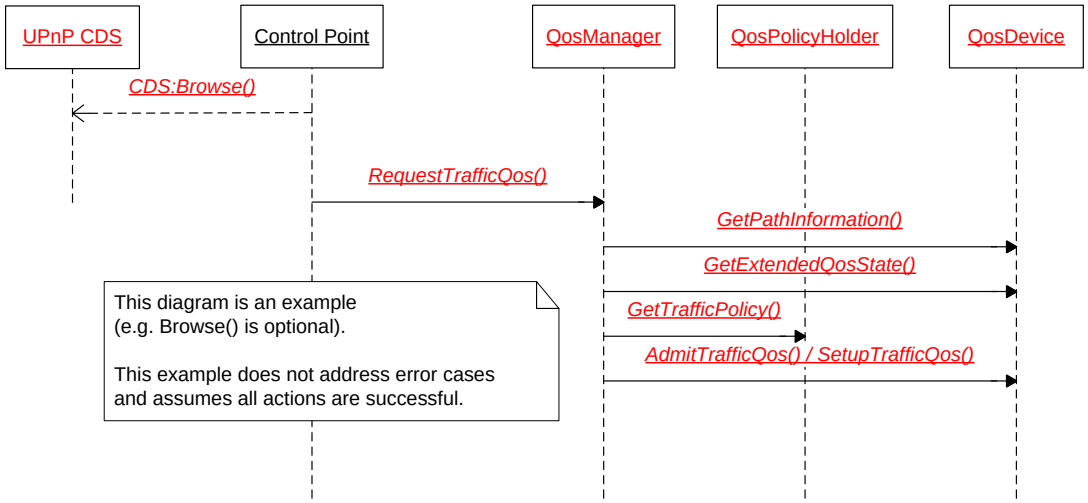

Figure 2: An Example Interaction Diagram for *RequestTrafficQos()* action for prioritized QoS setup.

3.2.2. Parameterized QoS

This section describes the interaction between the services for a request for Parameterized QoS. Figure 3 illustrates the interaction for admission only, whereas Figure 4 includes the interaction for additional QoS Manager capabilities such as preemption and reporting of blocking streams.

In response to a request for Parameterized QoS, the QoS Manager identifies the *QosDevice* Services on the path. The QoS Manager also identifies the QoS Segments. Further details are provided in Chapter 6.

The QoS Manager attempts to admit the traffic descriptor to the *QosDevice* Services on the path of the traffic stream using the most preferred TSPEC in the list.

In admitting a *TrafficDescriptor* for a certain TSPEC, the QoS Manager decomposes the end-to-end TSPEC parameters to per-QoS Segment parameters. The QoS Manager subsequently invokes the *AdmitTrafficQos()* action on the *QosDevice* Service(s) on each QoS Segment. The *QosDevice* Service will in turn invoke the appropriate Admission Mechanisms to request resources on the underlying L2 Technology.

If admission fails for the *TrafficDescriptor*, then the *RequestTrafficQos()* action fails at any *QosDevice*, Service. If the *QosManager* Service supports reporting of blocking traffic streams and/or preemption and if the Control Point has requested the use of these capabilities using the *RequestExtendedTrafficQos()* action, then: (1) the QoS Manager gathers the information about the blocking traffic streams; (2) the QosManager invokes the preferred *QosPolicyHolder* Service to determine the importance order of these existing traffic streams in relation to the new traffic stream; (3) The gathered information is reported to a Control Point.

The Control Point can use this information to inform the user about the state of the network.

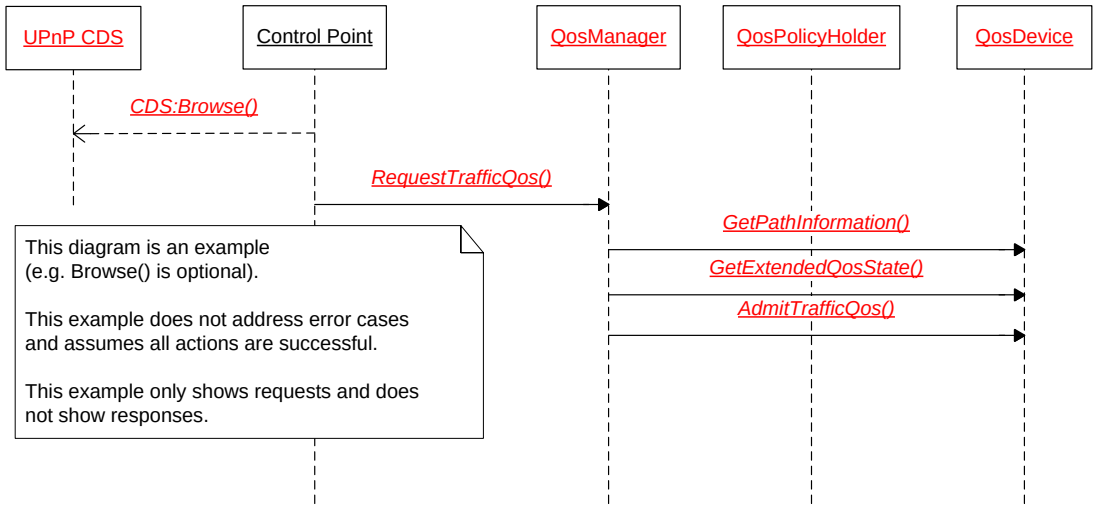

Figure 3: An Example Interaction Diagram for RequestTrafficQos() (without preemption).

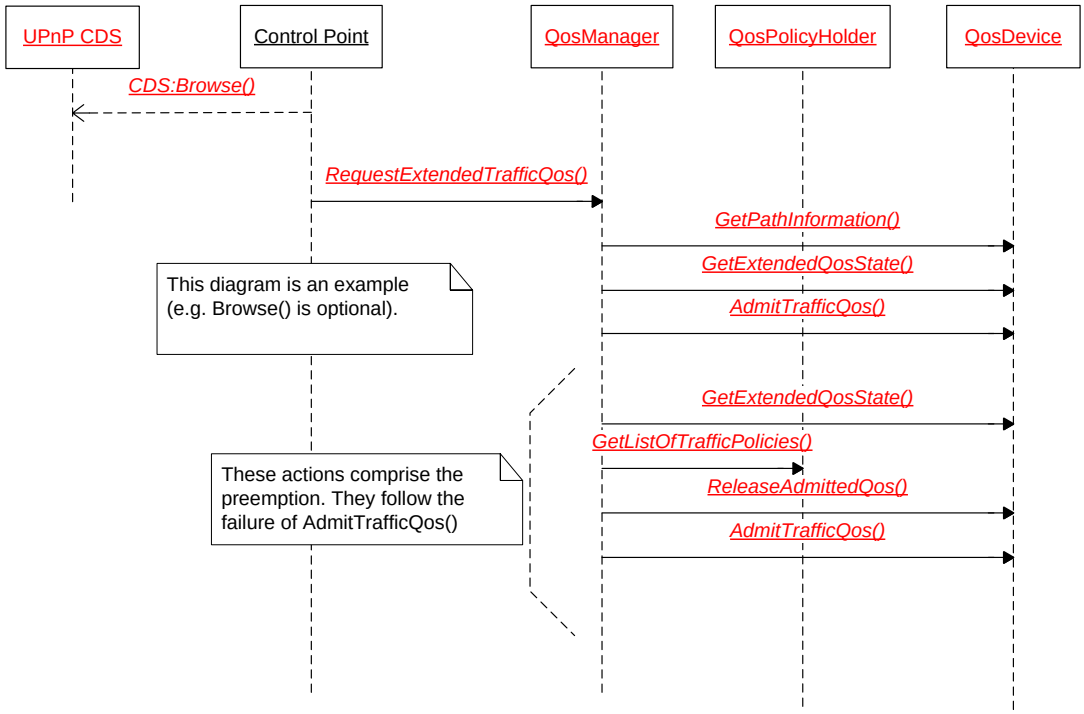

Figure 4: An Example Interaction Diagram for RequestExtendedTrafficQos() with preemption capability.

If preemption is requested, the QoS Manager determines which traffic streams can be preempted to provide sufficient resources for the least preferred TSPEC of the new traffic stream to be admitted on the network. The QoS Manager releases the QoS resources of the traffic streams it determined to preempt and retries admission of the new traffic stream.

Parameterized traffic streams contains a *finite* traffic lease time. The traffic lease time can be updated during the lifetime of the traffic stream. When the traffic lease time has expired, the *QosDevice* Service releases the

resources. Having a finite traffic lease time ensures that the system can return to a clean state if Control Points or QoS Managers that had set up QoS for certain traffic streams are no longer active.

3.2.3. Hybrid QoS

The setup of Hybrid QoS is similar to the setup of Parameterized QoS.

The difference is that on prioritized QoS Segments, QoS setup is done by the QoS Manager through the *QosDevice* Services actions *SetupTrafficQos()* or *AdmitTrafficQos()* action. The reason is that prioritized QoS Segments can have *QosDevice:1* Service [QD:1], *QosDevice:2* Service [QD:2], or *QosDevice:3* Services [QD:3]. Even if a device implements a *QosDevice:3* Service it does not necessarily mean that it supports parameterized QoS.

4. Key Concepts and Examples

This section provides examples for the key concepts of UPnP-QoS.

4.1. Interfaces and Links

Every *QosDevice* Service will have at least one interface which is defined as the point of interconnection between a device and a network. Incoming or outgoing traffic streams use an interface to get from or to the network. Within a *QosDevice* Service, Interfaces are identified by their *InterfaceId*. Example values for *InterfaceId* are "eth0" or "Wireless Network Connection". An interface is of a single technology such as Ethernet.

An interface connects the device to the network and thereby to other devices.

A *link* is an (possibly bidirectional) direct connection between two devices for data exchange. An interface can contain multiple links. In a device, a link can only belong to one interface. Different links can have different properties. For example, in a wireless network the link from the Access Point to one station can have a different signal level and different throughput than the link from the same Access Point to another station. Within an Interface, links are identified through their *LinkId*.

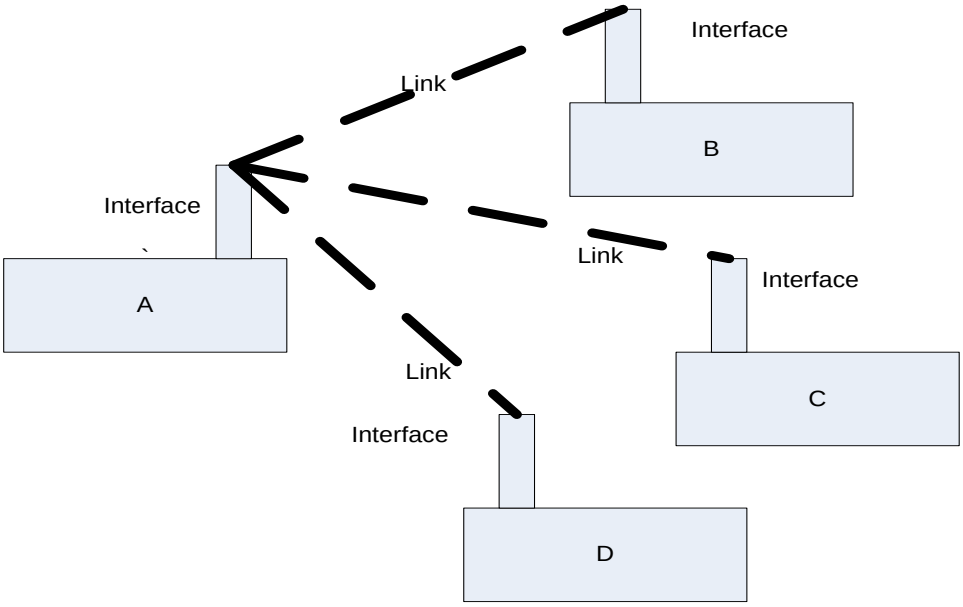

Figure 5: Example of Interfaces and Links. Device A has 1 interface that contains 3 links. Device B, C, and D each contain an interface with only a single link.

Prior to UPnP-QoS v3, the terms Interface and Link have been used interchangeably. UPnP-QoS v3 defines an interface container as the container for links.

4.2. Path Information

The path, as defined in [QM:3], can be determined on the basis of the path information supplied by the *QosDevice* Service. The path is ordered from source to sink by the sequence of *QosDevice* Services through which it passes.

The *QosDevice* Service reports a *PathInformation* structure that contains for each interface, the MAC addresses that have been observed on each of the link(s) that interface contains and whether links are actively bridged within the device.

UPnP-QoS is limited to QoS on a single subnet, therefore the following holds: An intermediate device is on the path if and only if it reports the MAC address of the path's source on a different link than the MAC address of the path's sink and these two links are bridged.

The path from source to sink can now be composed, by the QoS Manager, by sorting all of the devices in the order in which traffic flows starting with the source. This path information can be used to determine adjacency for purposes of invoking an admission request on a *QosDevice* Service.

4.2.1. Example – Bridge on the path

Consider the example in Figure 6. The *QosDevice* Service in this bridge reports that the bridge has one interface containing 4 links: p1, p2, p3, and p4. It reports that the source is on p1 and the sink is on p3. It also reports that p1, p2, and p3 are bridged by the same internal bridge but p4 is not (this is not illustrated in the figure). Since the source and the sink are on different link (p1 and p3 respectively) and the bridge between p1 and p3 is active, this bridge is on the path.

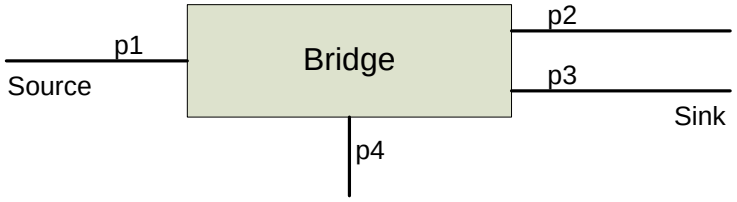

Figure 6: A bridge is on the path if and only if it reports the MAC address of the source on a different link than the MAC address of the sink and these two links are bridged.

4.2.2. Example – Device not on the path

Consider the example in Figure 7. In this example the laptop is not the source nor the sink of the stream. The *QosDevice* Service in the laptop reports that the laptop has 2 interfaces, each containing a link: p1 (e.g., its wireless interface) and p2 (e.g., its wired interface). It reports that the source is on p1 and the sink is on p2. However, p1 and p2 are not bridged by the laptop. The laptop is therefore not on any path as an intermediate device.

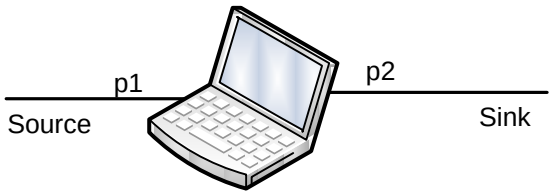

Figure 7: Laptop is not bridging its interfaces and therefore not on the path

4.2.3. Example – Path Determination

In this example we perform the complete path determination in a larger example: There are three devices (Source, Sink, and Dev 3) and three bridges (Bridge A, Bridge B, and Bridge C) in the network. They are depicted by hashed boxes in Figure 8. Every device also implements the *QosDevice* Service.

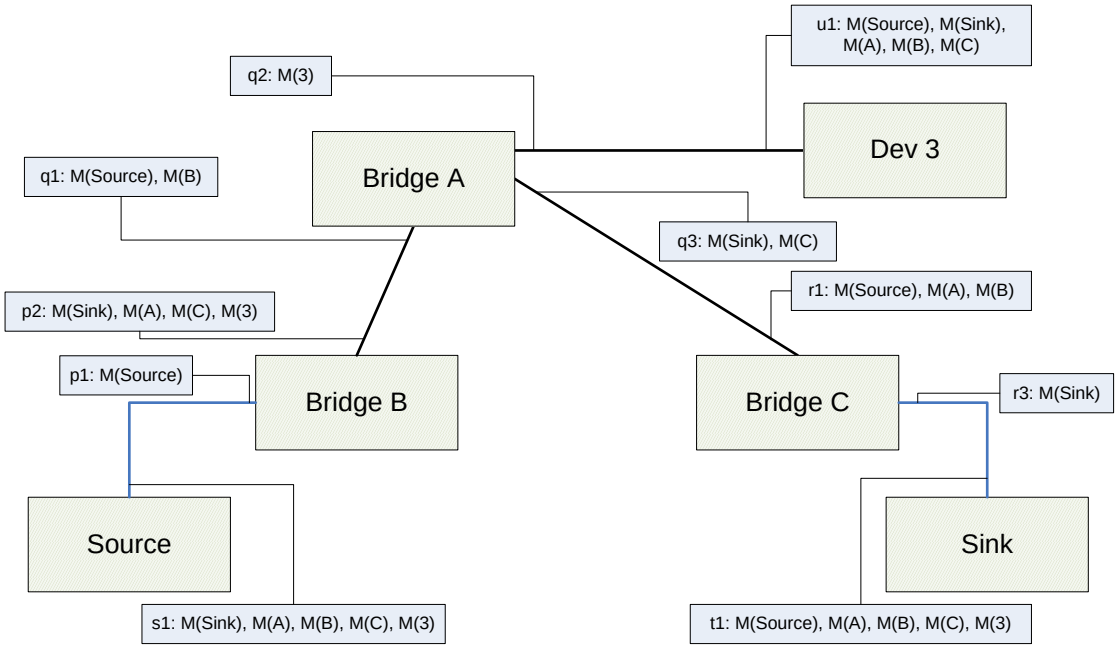

Figure 8: Example network for path determination

The blue solid boxes indicate what is reported by the *QosDevice* Service per link in the path information structure. The M(x) refers to the MAC address of x. For example "p1: M(Source)" means that the MAC address of the Source is reported on link p1.

As explained above, the bridges A, B, and C are part of the path from Source to Sink. For bridge A, the involved links are q1 and q3. For bridge B the involved links are p1 and p2. Finally for bridge C, the involved links are r1 and r3.

Note that Bridge A is further downstream than Bridge B (written B < A), because for Bridge A, both Bridge B and the Source are on link q1. Also note that B < C, this is because for Bridge C, both Bridge B and the Source are on link p1. Finally note that A < C, because for Bridge C, both Bridge A and the Source are on link p1. The bridges can now be sorted into B < A < C and the path is: Source – B – A – C – Sink.

In the example every device implements a *QosDevice* Service and therefore the path could be completely determined because each *QosDevice* Service is supplying path information. When there are bridges without a *QosDevice* Service the complete determination of the devices path is not possible from a UPnP-QoS perspective and the QoS Manager will consider just the parts of the path that it has determined.

4.3. QoS Segments

A QoS Segment is the (sub)set of *QosDevice* Service's interfaces that are under a single Admission Mechanism. The path from a source to a sink is composed of zero (this is the legacy case of non-UPnP-QoS v3 *QosDevice* Services) or more QoS Segments. When a *QosDevice* Service has more than one interface and those interfaces are in different QoS Segments, each interface is considered separately for QoS setup. A QoS Segment is needed for an Admission Mechanism that manages resources that are contained on the Segment and not solely on a *QosDevice* Service. An example is a technology that has a central coordinator that manages the network resources (e.g., transmission opportunities (TXOPs)) for that technology and is only accessible via signaling on that technology segment and not at layer 3 and above. A QoS Segment could contain ingress and egress interfaces when a stream is instantiated on that QoS Segment. The discovery of the path from source to sink produces an ordered list of *QosDevice* Service's interfaces through which the traffic passes. Each of these *QosDevice:3* Services' interfaces identifies the QoS Segment on

which it resides. The identifier is called the *QosSegmentId*. It is unique for each QoS Segment on the network and is identical for all *QosDevice* Service's interfaces on a given Qos Segment. The *QosSegmentId* is generated by each *QosDevice* Service by rules defined in the Technology Addendum.

The *QosDevice* Services on the path are queried for the *QosSegmentId* for each of their interfaces. The *QosSegmentId* is passed to a *QosDevice* Service when admission, update, or release is performed for a QoS Segment. The recipient *QosDevice* Service uses the *QosSegmentId* for purposes of managing Segment resources, in addition to managing any resources local to the *QosDevice* Service (e.g, buffers).

The following sections provide examples of QoS Segments.

4.3.1. Example – Simple QoS Segment

In Figure 9 a simple network consisting of four power line devices is drawn. The network contains three devices that contain the *QosDevice* Service and a fourth infrastructure device that does not contain a UPnP service and is indicated by the term "CCo".

According to the *QosSegmentId* formation rules (see [QDA:3]), the three *QosDevice* Services independently report the same *QosSegmentId*. It can therefore be concluded that the three devices are in the same QoS Segment. The "CCo" is also in that QoS Segment. However if there is no *QosDevice* Service on the "CCo"-device, this "CCo" cannot be managed directly by UPnP-QoS and its existence is therefore (at the UPnP-QoS level) irrelevant.

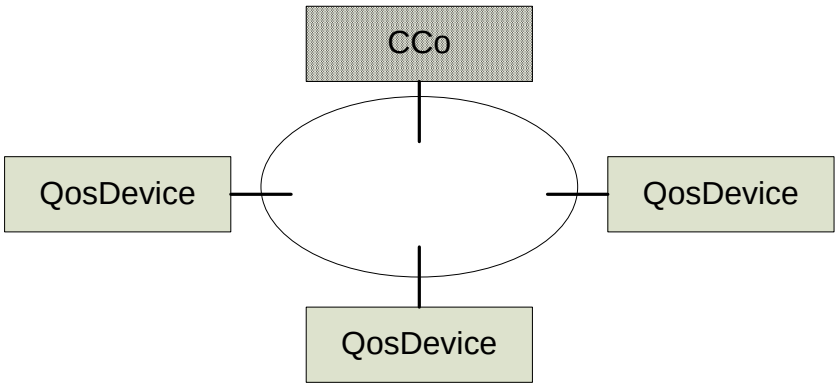

Figure 9: A simple network with one QoS Segment

4.3.2. Example – Multiple QoS Segments

In Figure 10 the network is extended with an additional Wi-Fi network. The QosDevice C reports two *QosSegmentId*s, one starting with 174 on one interface and one starting with 071 on the other interface. QosDevice D reports the same *QosSegmentId* (starting with 071). This network contains two QoS Segments. When a stream traverses both QoS Segments, both QoS Segments will require QoS setup via the QosDevice Services on the Segments. Thus QosDevice C will be setup for QoS twice, once for each interface. For example, suppose that QosDevice A is the source and QosDevice D is the sink. The path is: A then C (first the interface on the 174-Segment, then the interface on the 071 Segment) and then D

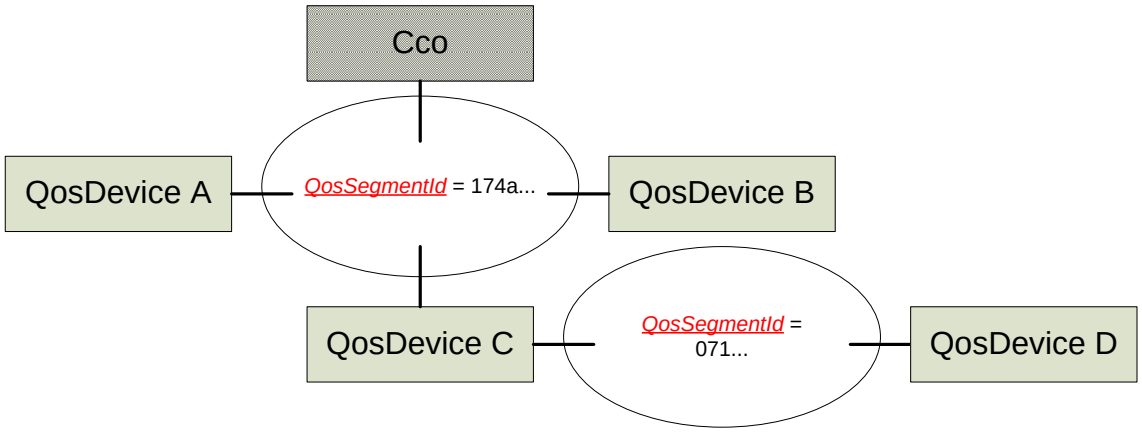

Figure 10: A network with two different technologies and two QoS Segments

4.3.3. Example – Homogeneous QoS Segment with L2 QoS Bridges

Another network is shown in Figure 11. This network consists of five devices and three bridges. The bridges are special bridges in that they also bridge L2 QoS setup requests; such a bridge is called, in this example, an L2Q bridge (for Layer-2-QoS Bridge). Every interface of every device is wired Ethernet. In the architecture, no distinction of links within the interface is made. For ease of explanation, we assume all devices implement the *QosDevice* Service.

Devices A, B, and C are end points in a QoS Segment bridged by an L2Q bridge and report the same *QosSegmentId*. Bridge 2 has four interfaces, two in the QoS Segment bridged by an L2Q bridge and a third and a fourth interface in the "006" Segment. Device D reports a *QosSegmentId* starting with "006" for Ethernet. Device D is therefore in another QoS Segment. Because Device D also connects to Bridge 2, the interface from Bridge 2 reports a *QosSegmentId* starting with "006" and the interface of Bridge 2 to Device D is not part of the L2Q QoS Segment with *QosSegmentId* "L2Q000000".

Bridge 3 and Device E are attached to the "006" Segment. Therefore Bridge 3 and Device E will report a *QosSegmentId* starting with "006" for Ethernet.

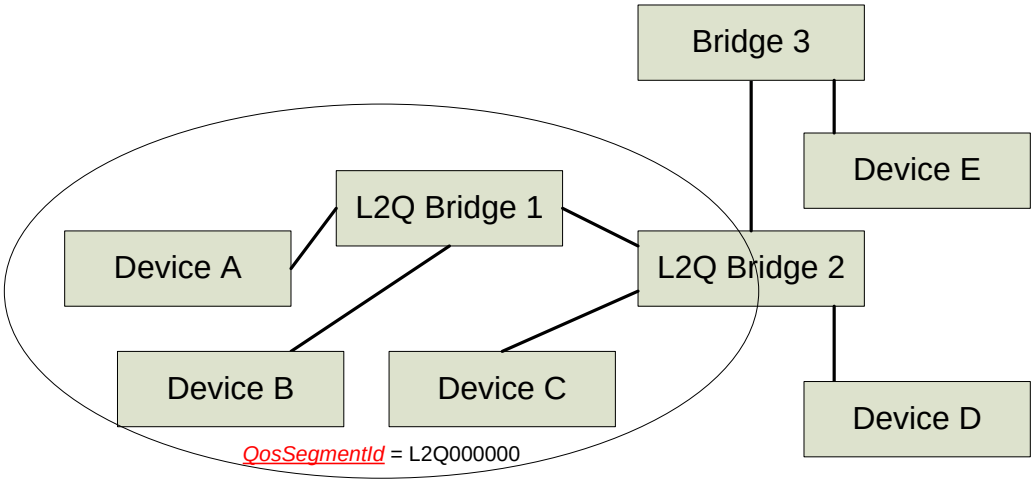

Figure 11: A network with Ethernet Layer-2-QoS bridges, L2Q-end point devices and legacy Ethernet bridges and legacy Ethernet devices

4.3.4. Example – Heterogeneous QoS Segment with L2 QoS Bridges

The fourth example is a network where different L2 technologies are bridged through Layer-2-QoS bridges. An example is shown in Figure 12, where L2Q-end point devices over Ethernet are bridged to L2Q-end point devices over MoCA. For ease of explanation, we assume all devices implement the *QosDevice* Service.

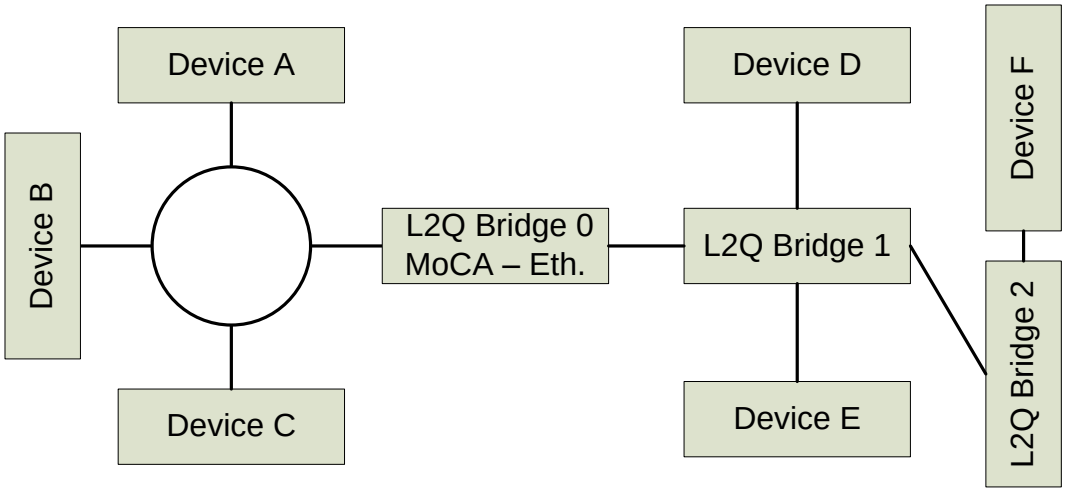

Figure 12: A network with all L2Q-end point devices but with different underlying technologies: MoCA (left hand side) and Ethernet (Right hand side), respectively

The L2Q Bridge 0 between MoCA and Ethernet (in the middle of the figure) bridges both traffic and L2Q QoS setup. A *QosDevice* Service implemented on the L2Q Bridge 0 (not shown) exposes the same *QosSegmentId* on both interfaces and hence does not divide the two sides into two QoS Segments. Therefore, the entire network in the figure will be treated as a single QoS Segment.

It is now demonstrated that all devices indeed are part of the same QoS Segment. On the left hand side, devices A, B, and C are connected via MoCA to a shared medium. The device A, B, and C are end point devices in a QoS Segment bridged by an L2Q bridge and report the same *QosSegmentId*. The devices D, E, and F are also devices in a QoS Segment bridged by L2Q Bridge 1 and L2Q Bridge 2 that have the same *QosSegmentId*..

4.3.5. *QosSegmentId* generation examples

The *QosDevice* Service Addendum [QDA:3] lays out the rules for creating a *QosSegmentId* for the technologies that were considered at the time of creating the UPnP-QoS specifications. The *QosSegmentId* is the same for interfaces that are connected to each other within the same QoS Segment but differs for interfaces in another QoS Segment, even if the interfaces use the same Layer 2 technology.

For Layer 2 technologies not described in the Addendum the following conditions are considered:

- Every *QosDevice* Service implementation on top of the Layer 2 technology has to generate a string.
- This string has to be identical for all interfaces of all *QosDevice* Service instances within the same QoS Segment.
- This string has to be unique to identify the QoS Segment from other QoS Segments in the same network.

There is no supporting technology provided by UPnP-QoS to do this. The *QosSegmentId* typically starts with 3 digits to indicate the IANA technology type and it is followed by a technology specific string. Typical examples are by using a network name, a string representation of a MAC address of the scheduler, the outcome of a Layer 2 leader election.

4.4. Adjacency of *QosDevice* Services

The concept of QoS Segment is introduced to hide the Layer 2 dependencies of QoS setup on the Segment, from the QoS Manager. A typical QoS Segment is illustrated in Figure 13 where on both the ingress as well as the egress end of the QoS Segment a *QosDevice* Service is located. The subject that is addressed in this section is: which service is responsible for performing (which part of) the Layer 2 QoS setup.The Layer 2 protocols define the capabilities of the devices.

If, for example, the Layer 2 technology for this QoS Segment requires the sink and not the source within the QoS Segment to setup QoS, then *QosDevice* Service B is able and needs to perform the Layer 2 request for a traffic stream that flows from A to B. Similarly the *QosDevice* Service A is able and needs to perform the Layer 2 request for a traffic stream that flows from B to A. For other Layer 2 technologies, it could be the source but not the sink that has to setup QoS, either the source or the sink that sets up QoS, or the source and the sink together that have to setup QoS.

The adjacency determination is performed by the QoS Manager and used to signal a *QosDevice* Service that there are adjacent *QosDevice* Services within the Segment. A *QosDevice* Service can use the adjacency information to avoid the situation where two *QosDevice* Services would accidentally reserve the resource twice, without requiring *QosDevice* Service to search for other *QosDevice* Services and without making a QoS Manager dependent on the specifics of Layer 2 technologies.

The QoS Manager supplies an input parameter to the *QosDevice* Service that contains two Boolean variables. The first Boolean variable *QDUpstream* indicates whether there exists another *QosDevice* Service further Upstream for the stream *within* the same QoS Segment. The second Boolean variable *QDDownstream* indicates whether there exists another *QosDevice* Service further Downstream for the stream *within* the same QoS Segment. The invoked *QosDevice* Service can make a determination, based on the Admission Mechanism that it implements, of the correct action for resource management; the QosDevice Addendum provides details for specific Admission Mechanisms.

Consider a traffic stream that enters the QoS Segment at A and leaves the QoS Segment again at B as in Figure 13. The *QosDevice* Service A is informed that there exists a *QosDevice* Service further downstream but not further upstream. Also *QosDevice* Service B is informed that there exists a *QosDevice* Service further upstream but not further downstream. If the Layer 2 requires the sink in the QoS Segment to setup QoS, then *QosDevice* Service B is responsible for performing the Layer 2 request for a traffic stream that flows from A to B and therefore requested to do so. In this case *QosDevice* Service A does not perform any action.

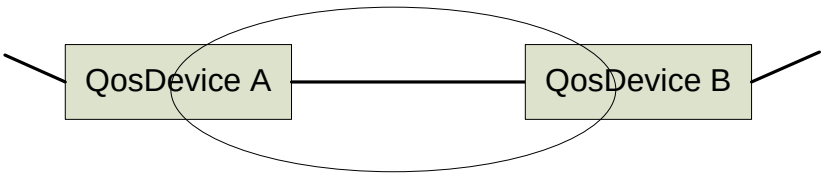

Figure 13: QoS Segment with two *QosDevice* services

Now consider the situation of Figure 14, where the sink in the Segment does not implement a *QosDevice* Service. In this case, the *QosDevice* Service A is informed that it has neither downstream nor upstream neighbors within the same QoS Segment. If the Layer 2 requires the sink in the Segment to setup QoS and does not allow the source to do so, then no QoS for the traffic stream from A to B can be requested.

However, if the Layer 2 merely prefers the sink in the Segment to setup QoS, but allows the sources to also do so, the *QosDevice* Service A will perform the setup because there is no other capable device.

The [QDA:3] document describes the requirements for certain technologies. They could be helpful to the reader as examples in understanding the general theory.

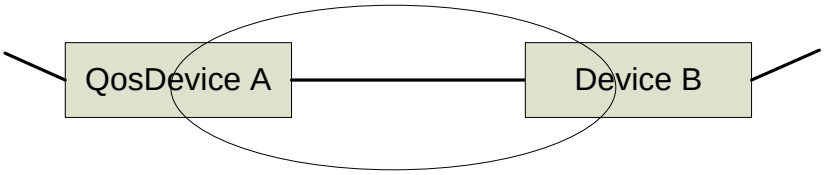

Figure 14: QoS Segment with only one *QosDevice* Service

The rules for the QoS Segment that are applied by the *QosDevice* Service for Admission Mechanism invocation are summarized in Table 4-1. For example if the Admission Mechanism for which the *QosDevice* Service needs to setup QoS requires the sink to setup QoS, the initiating *QosDevice* Service is the most downstream *QosDevice* Service. This *QosDevice* Service invokes resource managment because it is the only *QosDevice* Service that receives "*QDDownstream* = **0"**.

Adm.Mech. Initiation	Initiating *QosDevice* Service	Characterization
Only by the sink	Most downstream *QosDevice* Service	*QDDownstream* = 0
Only by the source	Most upstream *QosDevice* Service	*QDUpstream* = 0
By source OR sink	Most downstream *QosDevice* Service	*QDDownstream* = 0
By source AND sink	Most upstream *QosDevice* Service	*QDUpstream* = 0
	Most downstream *QosDevice* Service	*QDDownstream* = 0.

Table 4-1 Overview of the Admission Mechanism invocation

See [QDA:3] for how the rules apply to certain Layer 2 technologies.

5. UPnP-QoS Services

UPnP-QoS defines three UPnP services. The services are the *QosPolicyHolder* Service, the *QosManager* Service and the *QosDevice* Service. This section provides an overview of these three services, respectively.

5.1. The *QosPolicyHolder* Service

5.1.1. Overview

The UPnP *QosPolicyHolder* Service is a repository of QoS policies for the UPnP network. These QoS policies can be configured by the user or a third party on behalf of the user to indicate the treatment of traffic on the UPnP-QoS network. The *QosPolicyHolder* Service provides a UPnP-defined interface for a QoS Manager to access the network QoS policies. The policy within the *QosPolicyHolder* Service can be populated through a UPnP-QoS defined configuration mechanism (see Section 5.1.6).

The main function of this service in a prioritized network is to judiciously allocate the use of traffic importance numbers by traffic streams so that traffic importance levels are not overused. In a prioritized system the traffic importance number is mapped into a priority and if a given priority were overused, it would essentially lose differentiation.

The other role of the *QosPolicyHolder* Service is to provide information to differentiate various traffic streams on the network by their importance for the purposes of Preemption.

The *QosPolicyHolder* Service also provides additional information to identify the source of the QoS policy.

In UPnP-QoS v3 a mechanism is introduced for the user to prefer a *QosPolicyHolder* Service. The *QosPolicyHolder* Service also plays a role in maintaining this preference over time while devices are turned on or off.

5.1.2. Traffic Stream QoS Policy Description

The QoS traffic policy consists of elements specific to QoS and elements to identify the source of the QoS policy.

The QoS specific elements are

- *AdmissionPolicy* (a string),
- *TrafficImportanceNumber* (an integer in the range of 0-7), and
- *UserImportanceNumber* (an integer in the range of 0-255).

The elements of the QoS policy that identify the source of the QoS policy were introduced in UPnP-QoS version 2. These are

- *PolicyHolderId* (a string containing UDN and ServiceId),
- *PolicyLastModified* (a string with a date),
- *PolicyModifyingUserName* (a string), and
- *PolicyHolderConfigUrl* (a string with a URL).

The value of the *AdmissionPolicy* string is ignored for the version 3 *QosPolicyHolder* Service and its value is always "enabled". A version 3 QoS Manager requests admission control for parameterized QoS on the applicable *QosDevice:3* Services.

The value *TrafficImportanceNumber* indicates a traffic stream's priority. The *QosDevice* Service maps *TrafficImportanceNumber* to a Layer 2 priority as per the Addendum [QDA:3].

The value *UserImportanceNumber* is used by the QoS Manager as the basis for admission policy decisions when the network resources are saturated. The *UserImportanceNumber* value is used by the QoS Manager in admitting new traffic streams by preempting the QoS for existing traffic streams so that traffic streams with higher *UserImportanceNumber* values are accommodated first. A value of 255 indicates the highest importance and a value of 0 indicates lowest importance.

The *PolicyLastModified* string indicates with a date value when the policy was last modified, while the *PolicyModifyingUserName* string identifies the user or other entity that modified the policy for the last time. The *PolicyHolderConfigUrl* string contains a URL that points to the policy configuration page of the device.

5.1.3. Multiple instances of the *QosPolicyHolder* Services

The *QosPolicyHolder* Service provides an interface to supply the QoS policy for a given traffic stream, described by the *TrafficDescriptor*.

In UPnP-QoS version 1, the assumption was that there is exactly one *QosPolicyHolder* Service in the network.

In UPnP-QoS version 2, the existence of multiple instances of the *QosPolicyHolder* Service is allowed. The Control Point could indicate a *QosPolicyHolder* Service of its choice to the *QosManager* Service.

In UPnP-QoS version 3, the user has the ability to prefer a *QosPolicyHolder* Service. From then on every QoS Manager uses that Preferred *QosPolicyHolder* Service to determine traffic policy for requests for hybrid or parameterized QoS. For Prioritized QoS, the behavior of UPnP-QoS version 2 is maintained.

The behavior of the QoS Manager in the presence of different versions of the *QosPolicyHolder* Service is described in 6.3. The requirements can be found in the [QM:3].

5.1.4. Preferred *QosPolicyHolder* Service

In UPnP-QoS version 3, the user has the ability to prefer a *QosPolicyHolder* Service. From then on every QoS Manager uses that Preferred *QosPolicyHolder* Service to determine policy for requests for hybrid or parameterized QoS.

In order to select a preferred *QosPolicyHolder* Service, a control point invokes the *QPH:SetAsPreferred()* action with *SelectAsPreferred* = **1** parameter on the *QosPolicyHolder* Service it wants to make the preferred *QosPolicyHolder* Service. The *QosPolicyHolder* Service will then ensure that this preference is maintained in the network even when some devices are turned on or off. Any control point can also revoke the earlier preference of a *QosPolicyHolder* Service by invoking the *QPH:SetAsPreferred()* action with *SelectAsPreferred* = **0** parameter on any active *QosPolicyHolder* Service (i.e., not necessarily the Preferred *QosPolicyHolder* Service) in the network.

A non-preferred *QosPolicyHolder* Service will still respond to UPnP requests for QoS policy.

5.1.5. Maintaining the Preference of a *QosPolicyHolder* Service

This section describes how to maintain the preference of a *QosPolicyHolder* Service in a network where devices tend to come and go.

When a *QosPolicyHolder* Service is selected as preferred with the *QPH:SetAsPreferred()* action or when a Preferred *QosPolicyHolder* Service is rebooted then it performs the Preferred *QosPolicyHolder* Service propagation mechanism which is described below.

In this propagation mechanism process, the *QosPolicyHolder* searches for instances of the *QosDevice* Service on the network and invokes the *QD:SetPreferredQph()* action on each of them. This action needs to be invoked only once per advertised *QosDevice* Service. The *QosDevice* Service stores this information persistently across reboots. As soon as the action fails on a *QosDevice* Service with "Invalid Preferred QosPolicyHolder service" error, the *QosPolicyHolder* Service completes the propagation mechanism process and knows that it is no longer the Preferred *QosPolicyHolder* Service. If the *QD:SetPreferredQph()* action fails on a *QosDevice* Service for reasons other than those stated above then the *QosPolicyHolder* Service is still considered preferred.

5.1.6. Configuring the *QosPolicyHolder* Service

The policy in the *QosPolicyHolder* Service can be configured through a set of UPnP actions. The underlying model is that an ordered list of policy rules are maintained and used while responding to *GetTrafficPolicy()* and *GetListOfTrafficPolicies()* actions. To determine the policy, the *QosPolicyHolder* Service evaluates the list in order until there is a match.

The *AddQphPolicy()* action allows to insert a rule in a certain position in the table. The rule provides the filter conditions, the *TrafficImportanceNumber* and *ImportanceNumber*. The *ImportanceNumber* is a static priority that is used by the *QosPolicyHolder* Service to derive a relative *UserImportanceNumber* (see [QPH:3]).

The evented state variable *PolicyVersion* enables tracking the changes to the policy database. This allows a Control Point to detect asynchronous changes to the policy database.

5.2. The *QosManager* Service

The *QosManager* Service defines a set of actions for a Control Point to setup, release, and update the Quality of Service for a traffic stream. The *QosManager* Service performs those activities over the entire path of the traffic stream by making use of the *QosDevice* Services in the network. To control those *QosDevice* Services, a UPnP Control Point is needed. The behavior of this Control Point is driven by the actions invoked on the *QosManager* Service. This Control Point is called QoS Manager even if there is no exposed *QosManager* Service.

5.2.1. Overview

The QoS Manager is responsible for managing QoS assigned to various traffic streams. To fulfill its role, the QoS Manager acting as a Control Point discovers and invokes actions advertised by *QosDevice* Services and the *QosPolicyHolder* Services on the network.

The main function of the *QosManager* Service in a prioritized network is to request, for a traffic stream, the *TrafficImportanceNumber* value from the appropriate *QosPolicyHolder* service on the network. Based on the *TrafficImportanceNumber* returned from the *QosPolicyHolder* service, the QoS Manager configures the relevant *QosDevice* Services to assign the appropriate priority to the traffic stream.

In a network with admission control, the role of the QoS Manager is to admit a traffic stream to the network on the basis of the traffic stream's requirements from the traffic specification (TSPEC). The Control Point can supply a list of TSPECs with which the traffic stream can operate so that if one TSPEC cannot be admitted another can be attempted. For example two TSPECs can be provided for certain TV content: one for High Definition, and one for Standard Definition.

If admission fails, the QoS Manager performs preemption provided the preemption feature is implemented by the *QosManager* Service, requested by the Control Point and allowed by the *QosPolicyHolder* Service.

5.2.2. Behavior

When a QoS Manager completes setting up QoS for a traffic stream, all *QosDevice* Service(s) in the network path will have stored the traffic descriptor (more details are provided in the *QosDevice* Service section). Any QoS Manager can query the *QosDevice* Service(s) and determine the state of a traffic stream. This enables a *QosManager* Service to operate without maintaining any state.

Some *QosManager* Services implement the *RequestExtendedTrafficQos()* and *UpdateExtendedTrafficQos()* actions which support additional capabilities such as reporting of blocking streams and preemption. A Control Point can determine whether a *QosManager* Service implements the features "reporting of blocking streams" and "preemption", by invoking the *QM:GetQmCapabilities()* action.

A Control Point invokes the *QM:RequestTrafficQos()* or *QM:RequestExtendedTrafficQos()* actions to setup QoS by providing (among other things) a *TrafficDescriptor*.

The Control Point uses the *RequestedQosType* parameter to indicate that the Control Point wants to request parameterized QoS, or hybrid QoS (parameterized where available, prioritized where parameterized is not supported). If the *RequestedQosType* parameter is omitted, then prioritized QoS is requested.

The Control Point can also request that the *QosManager* Service reports which traffic streams are blocking admission or ask for preemption of the less important traffic streams in the case of failure to admit the requested traffic stream.

If a Control Point indicates a *QosPolicyHolder* Service of its choice and requests for prioritized QoS, then the *QosManager* Service will retrieve the *TrafficImportanceNumber* from the indicated *QosPolicyHolder* Service.

For requests for parameterized QoS, the QoS Manager determines the network path, the QoS Segments and will admit the traffic stream at all appropriate *QosDevice* Services.

The *QosManager* Service provides an interface (the *RequestTrafficQos()* and *RequestExtendedTrafficQos()* actions) for a Control Point to request QoS for a traffic stream identified by a *TrafficDescriptor*. The *QosManager* Service will return the *TrafficDescriptor* (in addition to other parameters) which will contain additional information such as the TSPEC that is currently active. If the *QosManager* Service is not successful, then it returns a corresponding error code.

Lastly, the *QosManager* Service provides an action for removing the QoS for a traffic stream.

5.2.3. Update the QoS reservation

The *QosManager* Service supports actions to update QoS of existing streams. If the update fails because the resources are not available, then the *QosManager* Service maintains the stream's current reservation.

In general, but specifically for update, it is the Control Point's responsibility to ensure that resources are requested before they are used and resources are no longer used before released. This is especially relevant when using Layer 2 Technologies that do not police actual resource usage.

5.3. The *QosDevice* Service

5.3.1. Overview

The *QosDevice* Service is implemented in a source, sink or intermediate network device. A QosDevice Service is responsible for managing the resources in the device. The *QosDevice* Service can also be responsible for managing and reporting the resources for other devices in the same QoS Segment.

5.3.2. Behavior

The *QosDevice* Service provides an interface for Control Points to execute actions on the device to admit traffic streams to setup QoS, to query the QoS capabilities and state of the *QosDevice* Service, to remove a traffic stream's QoS, and to register for events that the *QosDevice* Service generates. Typically, a QoS Manager acts as a Control Point for *QosDevice* Service. The *QosDevice* Service has an interface to configure and report per interface rotameter observations.

A *QosDevice* Service exposes its QoS capabilities through the *GetExtendedQosState()* action. This action supersedes the v1-defined *GetQosDeviceCapabalities()*, *GetQosState()*, and *GetQosDeviceInfo()* actions.

The *GetExtendedQosState()* action exposes QoS capabilities such as the type of Native QoS support, the maximum PHY bandwidth, IANA technology type, MAC address, *InterfaceId* and *AdmitCntrlNet*.

The run time QoS state is reported by returning a list of *TrafficDescriptor* structures for every traffic stream admitted or registered on the *QosDevice* Service..

The *AdmitTrafficQos()*, *ReleaseAdmittedQos()* and *UpdateAdmittedQos()* actions provide the core functionality to request admission, to release resources and to update the resource requests respectively. The *AdmitTrafficQos()* action sets up QoS. If *RequestedQosType* = 0 or absent, then Prioritized QoS is requested and a traffic stream is always admitted. For parameterized QoS Segments and *RequestedQosType* = 1 or 2, the *QosDevice* Service is requested to reserve the requested resource for the indicated TSPEC of the provided *TrafficDescriptor*. If this admission succeeds, the *QosDevice* Service reports success. If the resource cannot be reserved, then the request will fail. If a QoS Manager wants to establish prioritized QoS on QoS Segments where the admission fails, it therefore has to explicitly request Prioritized QoS.

The *UpdateAdmittedQos()* action can be used to request a change in reservation for an existing admitted traffic stream. The *UpdateAdmittedQos()* action has the following two properties:

- If the *UpdateAdmittedQos()* action fails, the current reservation will be maintained.
- If the *UpdateAdmittedQos()* action requests fewer resources than the original reservation (for example, decreasing the bandwidth requirement or increasing the maximum end-to-end delay requirement) then the request will be successful.

These two properties of the *QosDevice* Service allow a QoS manager to ensure that either an update requested by the Control Point succeeds or the current reservation is maintained in case of failure (see also Section 5.2.3 and Section 6.5.9).

For backward compatibility with version 1 and 2, the *SetupTrafficQos()* and *ReleaseTrafficQos()* actions are maintained. Their behavior is available through the *AdmitTrafficQos()* and *ReleaseAdmittedQos()* actions as well. The *SetupTrafficQos()* action of the QosDevice Service allows the QoS Manager to setup QoS associated with a particular traffic stream. The *ReleaseTrafficQos()* action of the *QosDevice* Service allows the QoS Manager to release QoS associated with a particular traffic stream.

The topology information known to the *QosDevice* Service is exposed via the *GetPathInformation()* action. The *GetQosDeviceInfo()* and *GetExtendedQosState()* actions allows the *QosDevice* Service to expose the IP addresses, port numbers and IP protocol of a particular traffic stream.

A *QosDevice* Service can collect information about network flows (rotameter) from its interfaces.

If a Control Point is interested in querying these rotameter observations, it first configures the service using the *ConfigureRotameterObservation()* action, then subsequently (e.g. a user initiated diagnostic session) request one or more observations using the *GetRotameterInformation()* action. The rotameter service has to be configured well before observations are requested (otherwise there could be insufficient data to provide diagnostic value).

The rotameter report contains counters that report the total traffic count for each device attached to a given interface or separate counters for each implemented priority queue. For parameterized QoS, *ROBitsParameterized* and *ROPacketsParameterized* provide the number of bits and packets related to parameterized QoS streams. There are also a number of parameters that are specific per traffic stream. These are the *StreamBitsTransmitted*, *StreamPacketsTransmitted*, *StreamPacketsReceived*, and *StreamPacketsDropped*.

These counters are useful for diagnostic purposes, i.e. ascertaining which device(s) on the network are most active (sending or receiving the most data), and therefore likely candidates for causing congestion. Because no distinction is made between bits sent or received in each counter, other UPnP-QoS methods are used to ascertain the source (versus the sink) of traffic (this can be done by simply querying the QosDevice for active *TrafficDescriptor* instances).

5.3.3. Configuring QoS

When the *QosDevice* Service action *AdmitTrafficQos()* or *SetupTrafficQos()* is invoked, the request is for a specific TSPEC. The *QosDevice* Service only attempts that TSPEC. The selection of different TSPECs is up to the QoS Manager. The *QosDevice* Service also does not perform preemption itself. If requested by the Control Point, preemption is performed by the *QosManager* Service (provided it supports that feature). The QoS Manager uses the *ReleaseAdmittedQos()* and/or *UpdateAdmittedQos()* action. .

5.3.4. Path Information

The *QosDevice* Service has an action that provides information regarding the devices in the network that are reachable through each of its active interfaces. This information is for a *QosManager* Service to correctly detect the topology, adjacency and the QoS Segments.

When there is a change in *PathInformation*, the *QosDevice* Service issues an event and sends the updated *PathInformation* variable in the body of the event.

5.3.5. Ancillary actions

In this section we describe four actions in the *QosDevice* Service that support the operations of the QoS system. The *SetL2Map()* action provides a *QosDevice* Service with a mapping from a *TrafficHandle* to a *Layer2StreamId*. With this mapping devices can filter on *Layer2StreamId* which is generally simpler than filtering on *TrafficId*.

The *VerifyTrafficHandle()* action is provided as an mechanism to verify whether a specific *TrafficHandle* is still valid on the *QosDevice* Service without searching in the complete list as returned by the *GetExtendedQosState()* action.

The *UpdateTrafficLeaseTime()* action allows to update only the traffic lease time of a *TrafficDescriptor* without doing a full *UpdateAdmittedQos()* action.

The *SetPreferredQph()* action supports the selection of a preferred *QosPolicyHolder* Service.

5.3.6. Events

The *QosDevice* Service can implement two evented state variables to keep track of the registered and admitted traffic streams: *MostRecentStreamAction* and *UnexpectedStreamChange*. The *MostRecentStreamAction* state variable contains counters that track whether the *SetupTrafficQos()* and *ReleaseTrafficQos()* actions are successfully invoked. The *UnexpectedStreamChange* state variable is a counter which is incremented after unexpected stream changes which could either be because unexpected things happen at the L2 Technology or when a traffic stream's resources are otherwise removed. Both these events are moderated to avoid flooding the network with repeated events. After receiving the event that indicates a change for *UnexpectedStreamChange*, a subscribed QoS Manager or other Control Point invokes the *GetUnexpectedStreamChanges()* action to obtain more information on what has happened.

6. System Operation

An overview of the operational flow of UPnP-QoS control messages is given below, more detailed information is provided in the related subsections.

1. Initiation of the QoS setup for a traffic stream – The *QosManager* Service requires a minimum set[2] of information from a Control Point to perform the QoS setup. The method used to gather this information for different usage environments is described in Sections 6.1 and 6.2.

2. Determination of Policy for the traffic stream – The QoS Manager determines the appropriate policy by requesting this information from the *QosPolicyHolder* Service. In some situations, the QoS manager will have to apply default policy.

3. Determination of *QosDevice* Services that have to be managed – Based on the source and destination information for the traffic stream the QoS Manager determines which *QosDevice* Services play a role in the transport of the traffic stream.

4. Configuration of *QosDevice* Service – the QoS Manager will interact with each applicable *QosDevice* Service to setup QoS for the traffic stream on the device. The device's setup will depend on the capabilities of the device and could include setting of packet handling priorities, other setup functions, allocation of resources, as well as Admission Mechanism signaling to request admission.

5. Return the results of the QoS setup to the Control Point invoking the *QosManager* Service – the success or failure of setup is provided to allow user feedback or other possible corrective actions.

6.1. Selection of a *QosManager* Service

The Control Point firsts select a *QosManager* Service. To find all *QosManager* Services, a Control Point performs an M-SEARCH for a *QosManager* Service embedded in *any* UPnP device type.

Control Points that do not select a *QosManager* Service, implement or provide a QoS Manager. There are zero or more *QosManager* Services in the network. These are known through standard UPnP discovery methods. The method for selection of a *QosManager:1* Service [QM:1] or a *QosManager:2* Service [QM:2] is not defined in the UPnP-QoS specification.

The selection of the *QosManager:3* Service typically depends on the capabilities of the *QosManager* Service. The *QM:GetQmCapabilities()* action is used to obtain the capabilities of the *QosManager* Service. The following capabilities are currently defined:

- The *QosManager* Service can perform admission on a First-Come-First-Served basis.
- The *QosManager* Service can, if capable, preempt existing traffic streams if their *UserImportanceNumber* is smaller than that of the new traffic stream to make room for the new traffic stream.
- The *QosManager* Service can report a list of existing blocking traffic streams so that the Control Point could perform some QoS management itself.

Every *QosManager:3* Service is capable of performing admission on a First-Come-First-Served basis.

---

[2] see the *TrafficDescriptor* Matrix in *QosManager* Service document [QM:3] for examples

6.2. Invoking the *QosManager* Service

When a *QosManager* Service is selected, its QoS setup operation is triggered either through the *QM:RequestTrafficQos()* action or through the *QM:RequestExtendedTrafficQos()* action. The *QM:RequestExtendedTrafficQos()* action was introduced in *QosManager:3* Service. The behavior of the *QM:RequestTrafficQos()* action is a subset of the *QM:RequestExtendedTrafficQos()* action.

It is recommended that Control Points that need to setup parameterized QoS for multiple traffic streams, invoke the *QM:RequestTrafficQos()* and/or *QM:RequestExtendedTrafficQos()* action serially. This reduces unnecessary competition for reservations. *QosManager* Services could also defer execution of the *QM:RequestTrafficQos()* and/or *QM:RequestExtendedTrafficQos()* action when they are still working on the completion of an earlier invocation.

The control point provides a *TrafficDescriptor* structure to the *QosManager* Service. This structure contains both the *TrafficId* which determines for which traffic stream QoS is requested and the TSPEC(s) that specifies resource requirements for the traffic stream. The *RequestedQosType* parameter is used to specify which type of QoS is requested (prioritized QoS, parameterized QoS, or hybrid QoS) as defined in Section 3.2.

The information provided in the *TrafficDescriptor* by the Control Point to the *QosManager* Service and the related process steps are described in the subsections. A generic process for any Control Point is described in Section 6.2.5. However, since streaming in UPnP AV is out-of-band in the specific case of an independent AV Control Point such a requester of QoS could not know all details of the specific traffic stream for which QoS is requested. The UPnP-QoS services add specific support to deal with this case by allowing an AV Control Point to supply an incomplete *TrafficId* and providing means for the *QosManager* Service to complete the *TrafficId*. This is described in Section 6.2.6.

6.2.1. Initiation of QoS Setup (I)

The *QM:RequestTrafficQos()* action sets up QoS. This provides the *QosManager* Service with a (partially complete) *TrafficDescriptor*. Upon completion of the *QM:RequestTrafficQos()* action, the *QosManager* Service returns the *TrafficHandle*, *NumPolicyHolders* and *UpdatedTrafficDescriptor* to the Control Point.

- The *TrafficHandle* is a unique identifier associated with that *TrafficDescriptor* to be used in all future interactions.
- The *NumPolicyHolders* value indicates the number of *QosPolicyHolder* Services that are available in the network if neither a preferred *QosPolicyHolder* Service exists nor a *QosPolicyHolder* Service was selected by the CP. A value other than "1" indicates that the default policies have been used by the *QosManager* Service. A value of "1" indicates that the policy provided by the preferred QosPolicyHolder Service or the Control Point selected *QosPolicyHolder* Service has been used or that exactly one *QosPolicyHolder* Service was found on the network.

6.2.2. Initiation QoS Setup (II)

The *QM:RequestExtendedTrafficQos()* action can also be used to setup QoS. Similar as with the *QM:RequestTrafficQos()* action, this provides the *QosManager* Service with a (partially complete) *TrafficDescriptor*. In addition to this, the *SelectedQmCapabilities* argument contains the options governing QoS setup. This contains the following information,

- The control point allows the *QosManager* Service to preempt (*Preemption*=**1**) or forbids the *QosManager* Service to preempt existing traffic streams (*Preemption*=**0**).
- The *QosManager* Service returns a list of blocking traffic streams if its admission was not successful（*ReportBlockingStreams*=**1**).

Upon completion of the *QM:RequestExtendedTrafficQos()* action, the *QosManager* Service returns the *TrafficHandle*, (an updated) *UpdatedTrafficDescriptor*, an integer *ResultedQosType*, and the *ExtendedTrafficQosInfo* including a (possibly empty) list of Blocking Traffic Streams to the Control Point.

- The *TrafficHandle* is a unique identifier associated with that *TrafficDescriptor* to be used in all future interactions.
- The returned *UpdatedTrafficDescriptor* is the one that was completed by the *QosManager* Service and was used in the setup of QoS.
- The *ResultedQosType* argument describes the result of the admission. It for example reports whether the end-to-end path passes through prioritized QoS Segments.
- The *ListOfBlockingStreams* in *ExtendedTrafficQosInfo* is populated when the *QosManager* Service was unable to admit the new traffic stream and the Control Point had requested a list of blocking streams.

The behavior of the *QM:RequestTrafficQos()* action is the subset of the *QM:RequestExtendedTrafficQos()* action with *Preemption*=**0** and *ReportBlockingStreams*=**0**.

6.2.3. Release of QoS Resources

The *QM:ReleaseTrafficQos()* action requests the *QosManager* Service to release QoS for the traffic stream that corresponds to the provided *TrafficHandle*. It is recommended that a *QosManager* Service process this invocation as soon as possible, as there could be ongoing QoS setup activities that benefit from the released resources.

6.2.4. Changing the QoS Setup

The *QM:UpdateTrafficQos()* and *QM:UpdateExtendedTrafficQos()* actions request a change in QoS. The behavior of the *QM:UpdateTrafficQos()* action is a subset of the *QM:UpdateExtendedTrafficQos()* action just like the behavior of the *QM:RequestTrafficQos()* action is a subset of the *QM:RequestExtendedTrafficQos()* action. The high-level properties of the update are that

1. Even if the *QM:UpdateTrafficQos()* or *QM:UpdateExtendedTrafficQos()* action fails, the reservation is maintained.
2. If the requested update results in a request of fewer resources, the *QM:UpdateTrafficQos()* or *QM:UpdateExtendedTrafficQos()* action succeeds.

6.2.5. Integrated Control Point

In the case of an integrated Control Point (a Control Point that is located in an endpoint), the UPnP-QoS architecture assumes Control Point will manage the binding between the traffic stream and the *TrafficDescriptor*. This means that the invocation of the *QosManager* Service is done by a Control Point that has knowledge of the setup of the transport method. The control point has to provide the *QosManager* Service with the complete *TrafficId* information and the TSPEC.

The integrated Control Point is present in the following applications:

- UPnP AV services, two box model: In the two box model there is a *MediaServer* device (MSD) and a Media Server Control Point. The device that hosts the Media Server Control Point (MSCP) is the initiator of the QoS setup. The MSCP will locate content on the MSD, retrieve its URI and start playback. The initiation of QoS Setup can be done after the establishment of an AV session for the traffic stream but before the content transfer starts.

- Generic case (not UPnP AV services): Any application is able to initiate QoS setup as long as it can provide the information needed by the *QosManager* Service.

6.2.6. Independent AV Control Point

This AV Control Point case refers to the situation where there is an AV Control Point that is not necessarily co-resident with the Media Render device (MRD) or the Media Server device (MSD). The AV Control Point contains a Media Server Control Point and a Media Renderer Control Point. This is referred to typically as the three box model. For further information refer to the UPnP AV architecture [AV].

In the three box model case, the AV Control Point initiating the *RequestTrafficQos()* action on the *QosManager* Service could be unaware of the complete *TrafficId*. The AV Control Point knows the UDN of the *MediaServer* device (typically the source) and the *MediaRenderer* device (the destination). The AV control point includes these addresses in its request.

The *QosManager* Service uses the *QD:GetQosDeviceInfo()* or *QD:GetExtendedQosState()* actions on the *QosDevice* Service instances in the *MediaServer* and the *MediaRenderer* device to query for the *TrafficId*.

The *QosDevice* Service in the *MediaServer* device returns the *SourceAddress* it will use for the traffic stream. This address could be different from the address the MediaServer uses for its UPnP communications. The *QosDevice* Service in the *MediaServer* also reports the port numbers and the *IpProtocol*. The destination reports the information from its side.

The UDN of the *MediaServer* and of the *MediaRenderer* device are typically not sufficient to uniquely identify the traffic stream in the *QosDevice* Services. Therefore the AV Control Point needs to include other AV specific information to help the *QosDevice* in the *MediaServer* and *MediaRenderer* devices to relate the QoS request to the AV requests. The items that can be provided are

- *MediaServerConnectionId* – this is the *ConnectionId* as provided by the *ConnectionManager* Service of the *MediaServer* device.
- *MediaRendererConnectionId* – this is the *ConnectionId* as provided by the *ConnectionManager* Service of the *MediaRenderer* device.
- *AVTransportInstanceId* – this is the *InstanceId* of the *AVTransport* Service on the *MediaServer* or *MediaRenderer* that is responsible for the AV transport of this traffic stream.
- *AVTransportUri* – this is the URI of the content which originates in the *ContentDirectory* Service.

To facilitate the identification by the *QosDevice* Service as many of the above parameters as possible have to be supplied.

6.2.7. Determination of QosBoundary Source and Destination

The QosManager will setup QoS on the network bounded by the *QosBoundarySourceAddress* / *QosBoundarySourceUuid* and *QosBoundaryDestinationAddress* / *QosBoundaryDestinationUuid*

If the Control Point provides the *QosBoundarySourceUuid* and not the *QosBoundarySourceAddress*, then the IP Address needs to be determined. It can be obtained from the *QosDevice* Service at the UPnP device with the provided UUID (likely a gateway).

If neither Address nor UUID format are provided, the QosManager assumes it manages the subnet on which it itself is located and determines the *QosBoundarySourceAddress* and *QosBoundaryDestinationAddress* if necessary.

6.2.8. Creation of the TSPEC (Traffic Specification)

The Control Point is responsible for providing the TSPEC (Traffic Specification) to the *QosManager* Service. These TSPEC parameters define the desired performance characteristics. The generation of the TSPEC parameters is left to the Control Point.

Parameters specific to content exposed via the AV *ContentDirectory:2* Service could have an associated *res@tspec* property.

6.3. Determination of Policy for the Traffic Stream

The QoS Manager communicates with the *QosPolicyHolder* Service to obtain the *TrafficPolicy*. In this section we describe how the QoS Manager determines the *QosPolicyHolder* Service that it needs to use.

6.3.1. Preferred *QosPolicyHolder* Service

In UPnP-QoS 3, the user can initially determine which *QosPolicyHolder* Service the user prefers and sets this preference through the *QPH:SetAsPreferred()* action. This *QosPolicyHolder* Service is called the Preferred *QosPolicyHolder* Service.

The *QosDevice* Services store this preference in persistent memory. The QoS Manager invokes the *QD:SetPreferredQph()* action to retrieve the *PreferredQphId* containing the **UDN** and **serviceId** of the *QosPolicyHolder* Service most recently set as preferred by the user and the *QphPreferenceCount* which quantifies this "recentness" of the user's selection. The QoS Manager now determines which *QphPreferenceCount* is the largest of all those reported by *QosDevice* Services on the network and uses the *PreferredQphId* associated to that number. If there are 2 (or more) *QosDevice* Services with the same *QphPreferenceCount* number but a different *PreferredQPHId* then there was a synchronization error. This error can occur in the rare case that two (or more) users simultaneously selected a new *QosPolicyHolder* Service and this error lasts until one user successfully reselects a *QosPolicyHolder* Service.

If the *PreferredQphId* value is "NULL", then the user did not prefer any *QosPolicyHolder* Service.

When the *QosPolicyHolder* Service associated with the *PreferredQphId* is not available or there are two *QosDevice* Services reporting a different *PreferredQphId* for the same largest *QphPreferenceCount*, the QoS Manager applies default policy (see Section 6.3.7).

6.3.2. CP-Indicated *QosPolicyHolder* Service

Since UPnP-QoS v2, the Control Point can indicate a *QosPolicyHolder* Service which is the *QosPolicyHolder* Service that has to be used. If a Control Point indicates a *QosPolicyHolder* Service, but that *QosPolicyHolder* Service is not currently available on the network, an error occurs and the Control Point's request for QoS fails.

6.3.3. Single *QosPolicyHolder* Service

In UPnP-QoS v1, if there is exactly one *QosPolicyHolder* Service on the network, that policy will be applied. If there are more *QosPolicyHolder* Services on the network or there is no *QosPolicyHolder* Service on the network the default policy is applied. The default policy is described in Section 6.3.7.

6.3.4. Priority Order of *QosPolicyHolder* Services for Prioritized QoS

For requests for Prioritized QoS, the following ordering is applied by the QoS Manager

1. CP-indicated *QosPolicyHolder* Service

2. Preferred *QosPolicyHolder* Service (if CP did not indicate *QosPolicyHolder* Service)

3. Single available *QosPolicyHolder* Service (if CP did not indicate *QosPolicyHolder* Service and Preferred *QosPolicyHolder* Service = "NULL")

If none of the above applies, default policy is used.

6.3.5. Priority Order of *QosPolicyHolder* Services for Parameterized QoS and Hybrid QoS

For requests for Hybrid QoS or Parameterized QoS, the following ordering is applied by the QoS Manager

1. Preferred *QosPolicyHolder* Service

2. CP-indicated *QosPolicyHolder* Service (if Preferred *QosPolicyHolder* Service = "NULL")

3. Single available *QosPolicyHolder* Service (if CP did not indicate QosPolicyHolder Service and Preferred *QosPolicyHolder* Service = "NULL")

If none of the above applies, default policy is used.

6.3.6. The *QosPolicyHolder* Service

The QoS Manager uses the *GetTrafficPolicy()* or *GetListOfTrafficPolicies()* actions of the *QosPolicyHolder* Service to retrieve the policy for the traffic stream defined by the information in the *TrafficDescriptor*. The *QosPolicyHolder* Service will provide the following information:

- *AdmissionPolicy* for prioritized QoS is compatible with UPnP-QoS versions 1 and 2. For all streams with *RequestedQosType* > 0, the *AdmissionPolicy* is ignored.

- *TrafficImportanceNumber* this parameter is used to determine the packet priority value (for prioritized and hybrid QoS requests) for the traffic stream. The QoS Manager provides it to the QosDevice Services, to allow priority tagging of the packets.

- *UserImportanceNumber* provides a relative ranking of the traffic stream's user importance compared to that of the other traffic streams. These numbers are used in the context of preemption (see Section 6.6)

- *PolicyHolderId* is a value that uniquely identifies a network device implementing the QosPolicyHolder Service.

- *PolicyLastModified* is a string that identifies the date/time when the policy on a device with the QosPolicyHolder Service was last modified.

- *PolicyModifyingUserName* is a string that identifies the user who last changed the policy

- *PolicyHolderConfigUrl* is a string containing the URL on the device that provides the QosPolicyHolder Service.

6.3.7. Default Policy

This paragraph describes the default policy...

The *TrafficImportanceNumber* default is based on the *TrafficClass*. This default mapping of Traffic Class to *TrafficImportanceNumber* is defined in the *QosManager* Service document.

The default *UserImportanceNumber* is 0, the lowest importance. The use of the same number for every traffic stream leads to First-Come-First-Served admission control.

6.4. Determination of *QosDevice* Services that have to be managed

To setup QoS for the traffic stream, the QoS Manager determines the devices that are directly involved in the transport of the traffic stream and the type of QoS setup that is requested. The Control Points (including *QosManager* and *QosPolicyHolder*) discover *QosDevice* Services embedded in *any* UPnP device type as it is expected that *QosDevice* Services can be added to any UPnP device (*MediaServer*, *MediaRenderer*, *InternetGateway*, *Basic*, etc.).

6.4.1. Configuration of QoS Devices

The configuration of the *QosDevice* Services is controlled by the QoS Manager. Depending on the particular requirements of the traffic stream and capabilities of the *QosDevice* Services on the path of the traffic stream, the QoS Manager determines the setup as described in the following sections.

6.4.1.1. Priority Setup Strategy

For prioritized QoS, configuring the traffic stream source device with the appropriate packet priority (*TrafficImportanceNumber*) is typically sufficient for setup of QoS. Upon successful completion of *QD:SetupTrafficQos()*, a source device implementing the *QosDevice* Service prioritizes the traffic associated with the *TrafficId*, according to the *TrafficImportanceNumber* on its output interface.

Intermediate devices are also configured with the traffic stream's *TrafficDescriptor*. Intermediate devices implementing the *QosDevice* Service with *PacketTaggingSupported* ="Yes" reprioritize the traffic associated with the *TrafficId* according to the *TrafficImportanceNumber* on their output interfaces irrespective of the priority on the incoming packets.

6.4.2. Path Determination

The QoS Manager uses the *TrafficId* information together with the response from *QD:GetPathInformation()* received from each *QosDevice* Service to determine which devices are on the path as described in Section 4.2. The information returned by *QD:GetPathInformation()* action can also be used to determine the applicable network interfaces.

The QoS Manager first determines the source and sink *QosDevice* Services by comparing the source and sink IP address in the *TrafficId* to the IP addresses of all discovered *QosDevice* Services available on the network.

The QoS Manager then identifies which other *QosDevice* Services are on the path with the help of the information returned by the *QD:GetPathInformation()* action (see the examples in Sections 4.2.1 and 4.2.2). An intermediate device is on the path if and only if it reports the MAC address of the source on a different interface than the MAC address of the sink and these two interfaces are bridged.

Finally the QoS Manager sorts the *QosDevice* Services based on the fact that for every *QosDevice* Service A and *QosDevice* Service B, *QosDevice* Service B is farther from the source than *QosDevice* Service A (A < B), if B reports that A and the source are on the same interface (see the example in Section 4.2.3).

6.4.3. QoS Segment Identification

When a traffic stream has to be setup, the QoS Manager determines the QoS Segments through which the traffic stream traverses.

The QoS Manager identifies QoS Segments by querying every *QosDevice* Service on the network via *QD:GetExtendedQosState()* action and comparing the *QosSegmentId*s of the different Interfaces. This process provides the QoS Manager with a list of all QoS Segments.

6.5. Admission Control

This section describes the Admission Control in further detail. In this section the procedure is completed by decomposing end-to-end requirements into per-QoS Segment requirements and setting up QoS on a Segment.

6.5.1. Decomposition of End-to-End Requirements into Per-QoS Segment Requirements

The QoS Manager sets up QoS on a per-QoS Segment basis but it needs to meet end-to-end requirements. This means that the QoS Manager needs to decompose the end-to-end requirements into per-QoS Segment requirements for some parameters. Other parameters apply globally.

Not every traffic stream has specific requirements on end-to-end delay, jitter or loss because the application can work with any finite delay, jitter or loss as long as these values are known to the application. The *QosManager* Service has to return the values for achieved upper bounds on end-to-end delays, jitter and loss.

As an example, suppose the Control Point has specified *E2EMaxDelayHigh* = 100 and there are two QoS Segments A and B. The QoS Manager could request a *QosSegmentMaxDelayHigh* = 50 at each QoS Segment. If both requests succeed, the requirement that *E2EMaxDelayHigh* = 100 is obviously met. The *QosDevice* Service returns the achieved *MaxCommittedDelay* and the *QosManager* Service returns the sum of the individual *MaxCommittedDelay* values to the Control Point as the achieved *E2EMaxDelayHigh*.

Another method is that, the QoS Manager does not include a specific per-QoS Segment requirement in its interaction with the *QosDevice* Service. The *QosDevice* Service minimizes delay when reserving resources for the traffic stream with the given TSPEC characteristics and to return the achieved value. The result could be that QoS Segment A achieves *MaxCommittedDelay* = 30 and QoS Segment B achieves *MaxCommittedDelay* = 60. The sum of these is clearly less than 100 and hence the requirement is met. Without further information a *QosDevice* Service provides the least *QosSegmentMaxDelayHigh* for its QoS Segment which could lead to an unnecessary burden on the resources.

A third method the QoS Manager can follow is to first investigate what the *QosDevice* Service can likely deliver. For this the QoS Manager invokes the *QD:GetExtendedQosState()* action which has the *TrafficDescriptor* structure as a input argument. From the returned *ProtoTspec* structure the QoS Manager can derive which maximum values for the *QosSegmentMaxDelayHigh* parameters would have worked if an admission was made at the time the *QD:GetExtendedQosState()* action was invoked. There is no guarantee that these values will work because the network state could have changed.

For some L2 technologies determining a *ProtoTspec* structure is straightforward because the values can be derived from the operating mode the L2 technology is in, the specific implementation or even the standard. But the determination of a *ProtoTspec* structure could for other L2 technologies amount to performing an admission and releasing any resources. A *QosDevice* Service on those latter L2 technologies will likely not return a useful *ProtoTspec* structure.

To support the decomposition of *E2EMaxDelayHigh* the parameter *E2EMaxDelayLow* can be used by the *QosManager* Service. The *E2EMaxDelayLow* parameter, if supplied by a Control Point, means that actual *E2EMaxDelayHigh* values below *E2EMaxDelayLow* are not needed. This is useful for Admission Mechanisms that would otherwise reserve with the smallest delay possible. For example, if the Control Point specifies *E2EMaxDelayHigh* = 1000 and *E2EMaxDelayLow* = 700, then this means that the *QosManager* Service has to achieve an *E2EMaxDelayHigh* of at most 1000 but it also means that it is not

necessary to achieve an *E2EMaxDelayHigh* below 700. If the QoS Manager can choose between achieving *E2EMaxDelayHigh* of 500 or 900, then 900 is preferable because it will typically pose a lower burden on the resources. Since 500 is also allowed, the *E2EMaxDelayLow* is therefore NOT the minimal delay.

The *E2EMaxDelayHigh* is returned to the Control Point. This specific information is computed by the *QosManager* Service after the admission at the *QosDevice* Services. If a new QoS Manager joins the network, it can only retrieve per-QoS Segment information from the *QosDevice* Services. It therefore has to calculate the actual *E2EMaxDelayHigh* itself.

6.5.2. Determination of adjacent *QosDevice* services within a QoS Segment

The *Resource* argument to the *QD:AdmitTrafficQos()* action and *QD:UpdateAdmittedQos()* action contains the adjacency of *QosDevice* Services. The Adjacency determination is explained in Section 4.4.

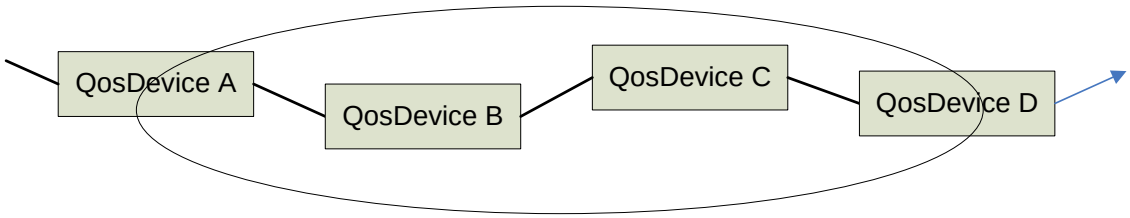

Figure 15: Example of Adjacent *QosDevice* services

As a first example consider Figure 15 with the traffic stream flowing from A to D. The QoS Manager provides the following information to the respective *QosDevice* Services.

QosDevice Service	*QosDevice* Service Downstream?	*QosDevice* Service Upstream?
A	1	0
B	1	1
C	1	1
D	0	1

We will now look at another example. Consider Figure 16 where device B and D do not implement the *QosDevice* Service. Here the QoS Manager provides the following information to the respective *QosDevice* Services.

QosDevice Service	*QosDevice* Service Downstream?	*QosDevice* Service Upstream?
A	1	0
C	0	1

There is a *QosDevice* Service downstream of *QosDevice* Service A (namely C) even though it is *not* a direct neighbor of A. For the determination of the values in the table, the devices that do not implement the *QosDevice* Service are irrelevant for the purpose of filling this table.

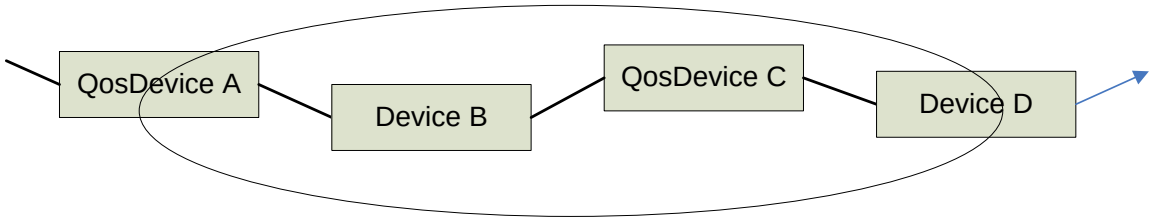

Figure 16: Example of Adjacent *QosDevice* Services. Note that only A and C are *QosDevice* Services.

6.5.3. Configuring *QosDevice* Services within a QoS Segment – release

When the QoS Manager releases QoS within a QoS Segment, it invokes the *QD:ReleaseAdmittedQos()* action for each *QosDevice* Service on the path within the QoS Segment. A *QosDevice* Service that is part of two (or more) QoS Segments has each of its interfaces managed separately, on a per QoS Segment basis. The *QosSegmentId* is used as input argument of the *QD:ReleaseAdmittedQos()* action to identify the QoS Segment for which the release of resources is requested. A QoS Manager can have more than one *QD:ReleaseAdmittedQos()* invocation outstanding at any time.

6.5.4. Configuring *QosDevice* Services within a QoS Segment

When the QoS Manager sets up QoS within a QoS Segment, it invokes the *QD:AdmitTrafficQos()* action for each *QosDevice* Service on the path within the QoS Segment. When the QoS Manager modifies an existing QoS reseveration within a QoS Segment, it invokes the *QD:UpdateAdmittedQos()* action for each *QosDevice* Service on the path within the QoS Segment. In the remainder of this section only the *QD:AdmitTrafficQos()* action is used. The interaction with the *QD:UpdateAdmittedQos()* action is similar.

A *QosDevice* Service that is part of two (or more) QoS Segments has each of its interfaces managed separately, on a per QoS Segment basis. The *QosSegmentId* is used as input argument of the *QD:AdmitTrafficQos()* action and *QD:UpdateAdmittedQos()* action to identify the QoS Segment for which the admission is requested. The reason for invoking the action once per QoS Segment, is that the per-QoS Segment Traffic Specification is possibly different for the involved QoS Segments. A QoS Manager can have more than one *QD: AdmitTrafficQos ()* invocation outstanding at any time.

The other input arguments of the *QD:AdmitTrafficQos()* action are explained above.

After the *QD:AdmitTrafficQos()* action completes, the *AdmitTrafficQosSucceeded* return argument contains the status report. If its value is *0* (success), the action was successful. If it is not zero, the *AdmitTrafficQosExtendedResult* return argument contains more information. In *AdmissionStatusDev* a reason code provides more details on why the admission failed if it failed for the device resources. In *AdmissionStatusNet* a reason code provides more details on why the admission failed if it failed for the network resources. If *AdmissionStatusNet* is “insufficient resources” then the network resources or device resources are currently insufficient to admit the traffic stream. A value “admission control not supported” means that this *QosDevice* Service does not support admission control for this traffic stream in this network configuration. This value can occur in several cases:

- The *QosDevice* Service is on a device that is intrinsically capable of performing admission control, but the QoS Segment this device is in lacks the supporting infrastructure and therefore the QoS Segment does not deploy admission control at this time.
- The *QosDevice* Service is on a device in two QoS Segments and supports admission control on one QoS Segment but not on the other QoS Segment and the request was for the QoS Segment in which it does not support admission control.

- The *QosDevice* Service is not at the boundary of a QoS Segment and it was asked to perform admission control because there was no suitable *QosDevice* Service at the boundary of the Segment (see Section 4.4 and Section 6.5.2).

In all cases, the QoS Manager concludes that no admission control can be performed on this QoS Segment and the QoS Manager has to rely on other QoS mechanisms such as Prioritized QoS.

6.5.5. Device resources managed by the *QosDevice* Service

Apart from network resources, the *QosDevice* Service can manage local device resources. The available device resources are accessible through the *QD:GetExtendedQosState()* action. The admission for a device resource is similar to the admission for a network resource, except that the *Resource* field is populated with the name of the device resource.

6.5.6. Collecting the results of all QoS Segments

When the QoS Manager has received a *Success* reason for every *QosDevice* Service on the path, the traffic stream is successfully admitted to the network and the *QosManager* Service can report a success to the Control Point. If there is a *QosDevice* Service where the traffic stream failed with a non-zero reason code the QoS Manager revokes the successful reservations it made at other *QosDevice* Services for this traffic stream.

The *ReasonCode* "insufficient resources" from a *QosDevice* Service could be caused by a transient limitation of resources. An example is that another QoS Manager was also in the process of admitting a traffic stream. A QoS Manager can retry admission up to three times, provided it obeys a random backoff time before retrying. Retries are not necessary, for example a QoS Manager that is confronted with denied admission at all *QosDevice* Services could decide it is not worth retrying at all.

Another reason for the indication "insufficient resources" is a persistent shortage of resources. The QoS Manager can proceed to attempt admission for a TSPEC with lower resource demands for the same *TrafficDescriptor*. For this to be possible, the Control Point has to supply an ordered list with multiple traffic specifications corresponding to different qualities of the same content in the original request.

While in the process of handling an admission request, a QoS Manager can have its successful reservations on one or more QoS Segments removed by another QoS Manager. This could mean that a Control Point has already decided to terminate the traffic stream and QoS is no longer needed or that another QoS Manager is performing preemption on this traffic stream (see Section 6.6). In both cases, the QoS Manager immediately ceases making admission requests and revokes its successful reservations.

If the *QD:AdmitTrafficQos()* action does not result in admission, then the return parameter list of blocking traffic streams provides useful information on traffic streams currently admitted on the QoS Segment. This list does not contain the *TrafficDescriptor* of such a traffic stream, but the traffic stream's *Layer2StreamId*. The *Layer2StreamId* value is the identification of the traffic stream at Layer 2. Not all Layer 2 technologies are able to report such stream identifiers. The QoS Manager needs to obtain the *TrafficDescriptor* for this traffic stream if it needs to report back to the Control Point (when *ReportBlockingStreams*=**1**) or to start the Preemption procedure (when *preemption*=**1**). In the case when the *TrafficDescriptor* cannot be obtained for a *Layer2StreamId*, then only the *Layer2StreamId* is returned.

A QoS Manager invokes the *QD:GetExtendedQosState()* action to obtain *TrafficDescriptor* to *Layer2StreamId* mappings. The *TrafficDescriptor* is typically only available at those *QosDevice* Services that are on the path of the traffic stream corresponding to that *TrafficDescriptor*. This means that the QoS Manager could have to invoke *QD:GetExtendedQosState()* action at all *QosDevice* Services on the same QoS Segment. If the *QosDevice* Service where the original *QD:AdmitTrafficQos()* action failed contains the *TrafficDescriptor* it will indicate this in the *TrafficDescriptorAvailable* field, this relieves the QoS Manager from searching for a *TrafficDescriptor*.

6.5.7. The *QosDevice* Service and the *QD:AdmitTrafficQos()* action

Following the invocation of the *QD:AdmitTrafficQos()* action, the *QosDevice* Service determines whether it is connected to the QoS Segment for which the admission is requested. Then the *QosDevice* Service determines on the basis of the adjacency of other *QosDevice* Services within the QoS Segment whether it is the appropriate *QosDevice* Service and has to perform the admission. If so the UPnP-QoS request is translated and issued as an appropriate Layer 2 request and the *QosDevice* Service returns an appropriate response to the QoS Manager.

The determination of whether this *QosDevice* Service is the most appropriate *QosDevice* Service to allocate the resources in the QoS Segment depends on the Layer 2 technology as defined in its Addendum. The table summarizes some cases.

Adm.Mech. Initiation	Initiating *QosDevice* Service	Characterization
Only by the sink	Most downstream *QosDevice* Service	*QDDownstream* = 0
Only by the source	Most upstream *QosDevice* Service	*QDUpstream* = 0
By source OR sink	Most downstream *QosDevice* Service	*QDDownstream* = 0
By source AND sink	Most upstream *QosDevice* Service	*QDUpstream* = 0
	Most downstream *QosDevice* Service	*QDDownstream* = 0.

If the Layer 2 technology requires the sink of a traffic stream to admit the traffic stream to the QoS Segment or the Layer 2 technology makes no choice, then the following happens. The *QosDevice* Service determines on the basis of the *Resource* argument to the *QD:AdmitTrafficQos()* action whether there exists a *QosDevice* Service farther downstream *within* the QoS Segment. If so, the *QosDevice* Service will not invoke the underlying Admission Mechanism for the traffic stream but rely on the *QosDevice* Service farther downstream to do this. If this *QosDevice* Service is the farthest downstream *QosDevice* Service, it will admit the traffic stream on the QoS Segment. This *QosDevice* Service is not necessarily located on the farthest downstream physical device within the QoS Segment, but it is the candidate defined by the technology addendum.

Analogously, if the Layer 2 technology requires the source of a traffic stream to admit the traffic stream, the *QosDevice* Service determines on the basis of the *Resource* argument to the *QD:AdmitTrafficQos()* action whether there exists a *QosDevice* Service further upstream *within* the QoS Segment. If so, it will not invoke the underlying Admission Mechanism for the traffic stream but rely on the *QosDevice* Service further upstream to do this. If this *QosDevice* Service is the furthest upstream *QosDevice* Service it will admit the traffic stream on the QoS Segment.

For a Layer 2 technology that requires both source and sink of a traffic stream to admit the traffic stream, both of the above will happen.

The next step the *QosDevice* Service performs is to translate the request into a Layer 2 admission request. This translation is specific to the Layer 2 technology. For example IP addresses can be converted to MAC addresses using ARP. Mappings from the UPnP-QoS Traffic Specification to certain Layer 2 technologies are provided in the *QosDevice* Service Underlying Technologies Addendum [QDA:3]. Certain UPnP-QoS traffic specification parameters are only to improve efficiency. Parameters not supported by a specific Layer 2 technology can be ignored.

When multiple Layer 2 requests are outstanding, the book-keeping to be performed by the *QosDevice* Service becomes more difficult. All *QosDevice* Service actions can be processed serially. *QosDevice* Service can process multiple action requests in parallel if they wish. .

6.5.7.1. Admission and the TSPEC

The *QosDevice* Service uses the TSPEC to determine whether the traffic stream can be admitted. The inputs in the TSPEC are the requirements from the application. The provided Quality of Service has to meet the requirements but this does not imply that it has to be exactly equal to the values in the TSPEC. For example, if the *DataRate* of 1 Mbps is requested, it is acceptable to provide a *DataRate* of 10 Mbps. Of course, providing more than what was requested could be inefficient.

For delay, the *QosSegmentMaxDelayHigh* parameter provides a requirement on the maximum delay over the QoS Segment. The minimum delay is not a parameter specifiable in the TSPEC and is assumed to be zero. In some L2 technologies this could lead to a difficult trade off because smaller delays can be realized but come with more inefficient use of the medium. In UPnP-QoS v3, no TSPEC parameter is provided that puts a requirement on the minimum delay. Such a requirement would lead to non-conformance if packets happened to be delivered faster than the requested minimum delay. To provide some guidance to implementers, the *QosSegmentMaxDelayLow* parameter is provided and is an indication similar to the *E2EMaxDelayLow* parameter. An actual QoS Segment delay that is less than *QosSegmentMaxDelayLow* is allowed but is not necessary.

6.5.8. The *QosDevice* Service and the *QD:ReleaseAdmittedQos()* action

Following the invocation of the *QD:ReleaseAdmittedQos()* action, the *QosDevice* Service determines whether it is connected to the QoS Segment for which the release is requested. Then it determines on the basis of the adjacency of other *QosDevice* Services within the QoS Segment whether it is the responsible *QosDevice* Service and has to perform the release. If so the UPnP-QoS request is translated and issued as an appropriate Layer 2 request and the *QosDevice* Service returns an appropriate response to the QoS Manager.

The determination of whether this *QosDevice* Service is the most appropriate *QosDevice* Service depends on the Layer 2 technology and the availability of neighbors that are better suited.

The next step the *QosDevice* Service performs is to translate the release request into a Layer 2 release request and perform the request. This translation is specific to the Layer 2 technology.

The *QosDevice* Service implementations can perform Layer 2 requests serially or in parallel. It is strongly recommended to process the *QD:ReleaseAdmittedQos()* action as soon as possible as it could be intended to free resources for other ongoing QoS set ups.

A successful invocation of the *QD:ReleaseAdmittedQos()* action with a populated argument *PreemptingTrafficInfo* results in an increase of the (evented) *UnexpectedStreamChange* state variable.

6.5.9. The *QosDevice* Service and the *QD:UpdateAdmittedQos()* action

Following the invocation of the *QD:UpdateAdmittedQos()* action, the *QosDevice* Service determines whether it is the responsible *QosDevice* Service and has to perform the update. This depends on the Layer 2 technology. The approach is similar as described for the *QD:AdmitTrafficQos()* action.

The next step the *QosDevice* Service performs is to translate the request into a Layer 2 admission request similar as in the implementation of the *QD:AdmitTrafficQos()* action. This translation is specific to the Layer 2 technology.

The *QD:UpdateAdmittedQos()* action returns successfully when the change has been made. This action is atomic to UPnP, but not necessarily atomic at Layer 2. Consequently the *QD:UpdateAdmittedQos()* action can fail if the resource allocation on the QoS Segment is changed while the action is processed at Layer 2.

When multiple Layer 2 requests are outstanding, the book-keeping to be performed by the *QosDevice* Service becomes more difficult. All *QosDevice* Service actions can be processed serially. *QosDevice* Service can process multiple action requests in parallel if they wish. .

6.6. Preemption

When the admission request of the traffic specification fails due to the lack of resources, the QoS Manager could attempt to free resources necessary to allow the new traffic stream to reserve them. The process of taking resources from existing admitted traffic streams is called preemption.

The process of preemption consists of three steps:

- identify the Blocking traffic streams (if the candidate traffic stream is not blocking, taking away its resources will not help in the admission of the new traffic stream),
- determine candidates for preemption, by applying policy to determine which Blocking traffic streams are less important and could be preempted,
- preempt traffic streams by releasing the QoS resources that are associated with identified traffic streams.

6.6.1. Identifying the Blocking Traffic Streams

The set of preemption candidates is initially populated by determining which traffic streams are blocking the admission of the new traffic stream. A list of those Blocking traffic streams is part of the *AdmitTrafficQosExtendedResult* return argument of the *QD:AdmitTrafficQos()* action. To be more precise, the *ListOfLayer2StreamIds* structure contains tupels of a *Layer2StreamId* value and a *TrafficDescriptorAvailable* Boolean value for Blocking traffic streams. A *Layer2StreamId* value first has to be converted to a *TrafficDescriptor* structure. If *TrafficDescriptorAvailable* equals true, the same *QosDevice* Service can provide the entire *TrafficDescriptor* structure. Otherwise, the QoS Manager has to find out the *TrafficDescriptor* structure corresponding to a *Layer2StreamId* value itself by querying other QosDevice Services on the QoS Segment. The *QD:GetExtendedQosState()* action provides the mapping between *TrafficDescriptor* structures and *Layer2StreamId* values as part of the *ListOfAdmittedTraffic* return argument. Certain underlying Layer 2 technologies are unable to determine a *Layer2StreamId* of a Blocking traffic stream. In that case, the QoS Manager has to fully rely on the *QD:GetExtendedQosState()* action to determine all traffic streams on the QoS Segment, which could include non-Blocking traffic streams.

6.6.2. Determining Candidates for Preemption

Next the QoS Manager ranks the individual traffic streams according to policy. The QoS Manager can do this by invoking the *QPH:GetListOfTrafficPolicies()* action on the list (determined in Section 6.6.1) of *TrafficDescriptor* structures that contains a *TrafficDescriptor* structure for every Blocking traffic stream and the *TrafficDescriptor* structure for the new traffic stream. The result of this step is precisely the return argument of the *RequestExtendedTrafficQos()* action of the *QosManager* Service when *ReportBlockingStreams*=**1** (but *Preemption*=**0**).

The list of traffic policies associates a certain *UserImportanceNumber* value *x* with the traffic descriptor for the new traffic stream. Only traffic streams with *UserImportanceNumber* value strictly smaller than *x* could be preempted. Between those traffic streams, the precise determination of which ones are preempted and which ones are not is dependent on the QoS Manager.

A simple example is to preempt traffic streams from the lowest value of *UserImportanceNumber* upwards until sufficient resources become available or the traffic stream has a *UserImportanceNumber* value of *x* or higher whichever is earlier. An example is shown in Figure 17.(a). A disadvantage of this approach is that the QoS Manager could end up unnecessarily preempting many traffic streams with only modest resource requirements before preempting the traffic streams with the larger resource requirements. Another possible method is to preempt the smallest set of traffic streams that provide sufficient bandwidth for which an example is shown in Figure 17 (b).

In the examples in Figure 17, the available bandwidth on the network is 3 Mbps. The existing traffic streams are indicated by ellipses. The new traffic stream is indicated by the square box and has a bandwidth requirement of 15 Mbps. The preemption approach has to work towards freeing 12 Mbps. The traffic streams are ordered by their *UserImportanceNumber* (UIN). Only the traffic streams with *UserImportanceNumber* < 50 can be preempted. Approach (a) preempts 3 traffic streams and frees 4+2+9 = 15 Mbps, which is sufficient. Approach (b) preempts the smallest set of traffic streams and frees 12 Mbps. Finally approach (c) determines that *y* = 30 and preempts 2 traffic streams freeing 4+9 = 13 Mbps.

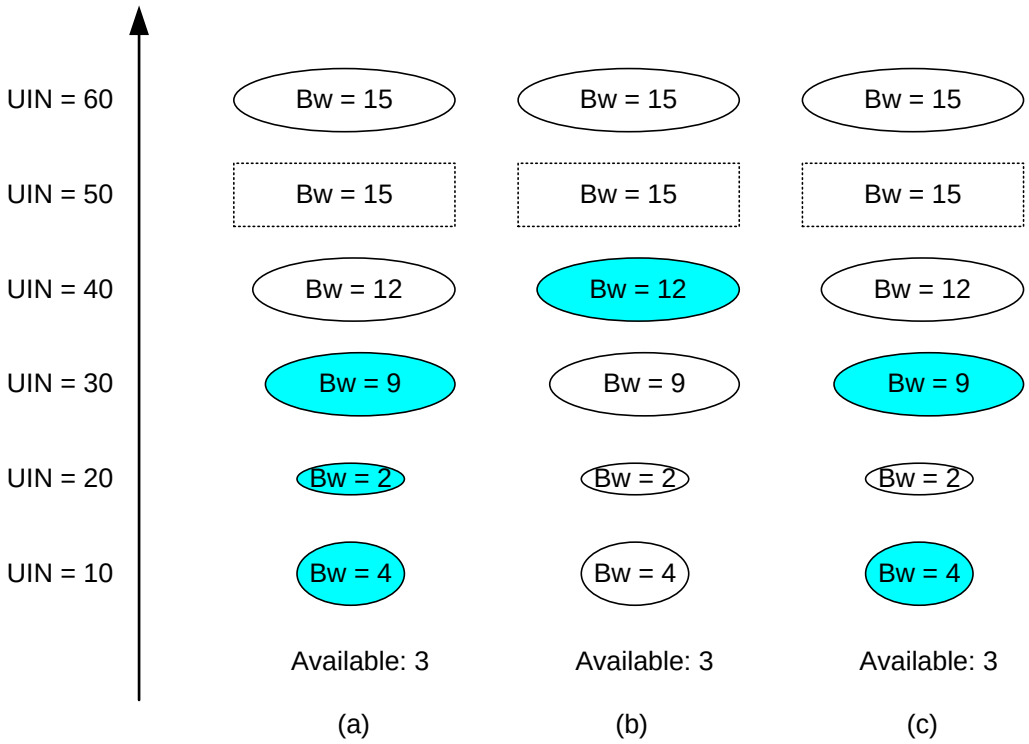

Figure 17: Examples of approaches to determine candidates for preemption

The following approach is suggested because its impact on existing traffic streams is relatively small and it follows *UserImportanceNumber* as much as possible. It is illustrated in Figure 17 (c). Calculate the highest value *y* of *UserImportanceNumber* for the traffic streams that would be preempted according to the previous approach, but do not actually preempt the traffic streams. Now preempt the smallest number of traffic streams with a *UserImportanceNumber* at most *y* that provides sufficient resources.

QoS Managers could have more advanced approaches and decide to modify (reduce) the Quality of Service of existing traffic streams rather than removing all resources.

In this section it is assumed that there is only a single blocking QoS Segment. Clearly there could be multiple blocking QoS Segments in which case other approaches could be preferable for reasons of book-keeping or user impact.

The *Critical* element of the *TrafficDescriptor* structure indicates,whether this traffic stream is critical and never to be preempted. This element is taken into account by the *QosPolicyHolder* Service while deciding the UIN of the stream and does not have to be inspected by the QoS Manager during preemption.

6.6.3. The Preemption and notification

The QoS Manager will now revoke the QoS resources associated to a traffic stream that has been selected for preemption. This basically means that the QoS Manager performs all the steps it would perform if the *QM:ReleaseTrafficQos()* action would have been invoked. In particular, the QoS Manager revokes the QoS resources end-to-end and not just at the blocking QoS Segments. The QosManager includes an input argument when invoking the *QD:ReleaseAdmittedQos()* actions to identify for which *TrafficDescriptor* the traffic stream is preempted. This allows QoS Managers or Control Points to identify the *TrafficDescriptor* of the preempting stream.

If the QoS Manager has decided to modify the Quality of Service of existing traffic streams, then it will have to be aware that the actual traffic has not yet changed and that this could remain the case for an indefinite period of time. Changing a resource reservation on a QoS Segment could result in a situation where admitted traffic streams are not conforming to their modified reservations.

Recall that as part of the release of the traffic stream's resources, the *UnexpectedStreamChange* state variable is incremented.

6.6.4. Re-Attempt To Admit the Traffic Stream

After all selected traffic streams are preempted, the QoS Manager performs a new attempt at admitting the new traffic stream as described in Section 6.5.4.

When the QoS Manager receives success for every *QosDevice* Service on the path, the traffic stream is successfully admitted to the network and the *QosManager* Service can complete the admission by reporting a success to the Control Point. If there is a *QosDevice* Service where the admission failed, the QoS Manager revokes the (successful) reservations it made at other *QosDevice* Services for this traffic stream.

The indication of failure could again be caused by a temporary shortage of resources. An example is that another QoS Manager was also in the process of admitting a traffic stream. The QoS Manager could retry admission on the QoS Segment up to three times. However, it does need to obey a random back off time before attempting again.

If a QoS Manager notes that its successful reservations (on other QoS Segments) have already been removed by another QoS Manager, then this could mean that a Control Point has already decided to terminate the traffic stream and QoS is no longer needed or that another QoS Manager is performing a preemption on this traffic stream. In both cases, the QoS Manager immediately stops its activities and will not retry admission.

If the admission still fails and *ReportBlockingStreams*=**1**, the QoS Manager returns the list of blocking streams.

6.7. Run time Operation

6.7.1. Traffic Lease Management and Link failures

The *TrafficDescriptor* includes a *TrafficLeaseTime* for the QoS resources allocated to the traffic stream. For parameterized and hybrid QoS requests, this lease time is bounded. The *QosDevice* Service revokes the resources allocated to the traffic stream after the *TrafficLeaseTime* has expired. The *QD:UpdateLeaseTime()* action provides a mechanism to update the traffic lease time with a single action.

The Control Point sets *UpdateTrafficLeaseTime* to "**1**" in the *TrafficDescriptor* and invokes the *QM:UpdateTrafficQos()* action to update the *TrafficLeaseTime* of the existing traffic stream.

6.7.2. Violation and Policing of the TSPEC

When a traffic stream is admitted, this admission was based on the active TSPEC that was provided as part of the *TrafficDescriptor* structure in the admission requests. The source device will transmit packets according to the *TrafficSpecification*. This section explains what could happen if the source device does not follow the *TrafficSpecification*.

If a source device sends less traffic than expressed by the *TrafficSpecification*, then network resources are unnecessarily allocated, but the traffic stream can continue.

The behavior for the case where a source device for a traffic stream sends more traffic than expressed by the *TrafficSpecification* is partially dependent on the underlying Layer-2 technology. Some Layer-2 technologies will still allow additional packets to be sent, provided there are sufficient resources. If there are insufficient resources, some Layer-2 technologies will police the traffic.

6.7.3. Being Preempted

In Section 6.6, we have described how a QoS Manager can preempt existing traffic streams. The preemption results in an increase of the evented *NumberOfUnexpectedStreamChanges* state variable of the impacted *QosDevice* Services.

A Control Point that is subscribed to the events of the *QosDevice* Service along the path of a traffic stream can then note that the traffic stream was preempted (see also Section 6.6.3). By querying the *QD:GetUnexpectedStreamChanges()* action it can identify whether it was its traffic stream. The Control Point can assume that all QoS resources for the traffic stream are freed in a clean manner. Following this, the Control Point can take action to stop the traffic stream, attempt to (re)admit the traffic stream at a lower quality, etc.

Finally, if a QoS Manager is preempting a traffic stream when it notices the traffic stream it wants to admit is preempted, then the QoS Manager will complete the preemption so that all preempted traffic streams are cleanly preempted and it will stop its attempt to admit the new traffic stream.

7. QoS Boundary Addresses

In the case of a scenario where the traffic either originates or terminates on the WAN (i.e. WAN-to-LAN or LAN-to-WAN traffic) the *QosBoundarySourceAddress* and/or *QosBoundaryDestinationAddress* fields of *TrafficDescriptor* can be used. These values could be used by the *QosManager* Service for such tasks as path determination. In such a scenario, the Traffic Identifier field of TrafficDescriptor enumerates the real source / destination IP address of the stream (whether it is on the LAN or the WAN). In case where the traffic originates on the WAN and ends up on the LAN, the Control Point will specify the *QosBoundarySourceAddress* or *QosBoundarySourceUuid* as the IP address or UDN of the *InternetGateway* Device. In case where the traffic originates on the LAN and ends up on the WAN, the Control Point will specify the *QosBoundaryDestinationAddress* / *QosBoundaryDestinationUuid* as the IP address or UDN of the *InternetGateway* Device.

This applies in particular to the *InternetGateway* Devices