# Inter-domain Controller (IDC) Protocol Specification

Version 1.1

February 9, 2010

# Table of Contents

# 1 Introduction

This document specifies the Inter-domain Controller (IDC) protocol for dynamically provisioning network resources across multiple administrative domains. The IDC architecture supports *dynamic networking*, the concept by which network resources (i.e. bandwidth, VLAN number, etc) are requested by end-users, automatically provisioned by software, and released when they are no longer needed. This contrasts more traditional “static” networking where network configurations are manually made by network operators and usually stay in place for long periods of time.

As the name suggests, the IDC protocol specifically addresses issues related to dynamically requested resources that traverse domain boundaries. In both the static and dynamic case there must be extensive coordination between each domain to provision resources. In the static case this requires frequent communication between network operators making manual configurations and can take weeks to complete depending on the task. In the dynamic case, the IDC protocol automates this coordination and allows for provisioning in seconds or minutes. Interactions between domains are handled using messages defined in the protocol.

The IDC protocol defines messages for reserving network resources, signaling resource provisioning, and gathering information about previously requested resources. These messages are defined in a SOAP [SOAP] web service format. Since all messages are defined using SOAP and XML, the protocol also utilizes a few external specifications for features such as security and topology description. Later sections in this document will indicate where external specifications are used. Also, the complete list of supported messages defined by the IDC protocol is contained within a Web Services Description Language (WSDL) file [WSDL]. This document describes the WSDL file and provides additional details on the information elements in each message. This document and others relating to the DICE IDCP are maintained at the DICE IDCP Control Plane web site: www.controlplane.net [CNTL-PLANE].

## 1.1 Goals and Requirements

The goal of the IDC protocol is to define the terminology, concepts, operations, and messages needed to dynamically provision network resources across multiple administrative domains.

### 1.1.1 Requirements

In meeting these goals the IDC protocol must address the following requirements:

- **Must securely communicate messages.** Security mechanisms that support authentication, authorization, and encryption must be factored into the protocol design. Security is vital to protecting the valuable network resources of communicating domains.
- **Must support multiple vendors and technology types.** The diversity of network equipment is an important consideration for an inter-domain protocol. The protocol design should be generic enough that its information elements are meaningful to configuring equipment made by different vendors and/or of differing technology type (i.e. Ethernet, MPLS, etc.).
- **Must provide information portable to other network services.** The dynamic allocation of network resources will be important to other services such as those dedicated to monitoring and measurement. The IDC protocol should utilize external specifications when it increases its ability to interoperate with other services without violating the other requirements.
- **Must allow for future extensibility.** Extensibility is important for supporting new user requirements as they arise in the future. It is also critical for supporting the dynamic provisioning of new network technologies as they become available.

### 1.1.2 Non-Goals

The following topics are outside the scope of the IDC protocol specification:

- **Defining an interface between an Inter-domain Controller and the Domain Controller.** The IDC architecture describes a domain specific service called the Domain Controller (DC) that manages and provisions local network resources. This document does not describe how information from IDC protocol messages is passed to the DC as it is domain-specific.
- **Defining security policy.** This document defines information elements used in IDC protocol messages that may be used to establish trust and make

authorization decisions, but it does not dictate how a domain uses that information to make such decisions.

- **Defining the information elements used to describe a domain's topology.** Topology description is specified using an external specification called the NMWG Control Plane Schema [NMWG-CP]. This document describes the aspects of that schema pertinent to its own information elements but is not an exhaustive description of the NMWG Control Plane definition. The DICE IDCP utilizes a specific version of this schema as defined in the reference [NMWG-CP] and on the IDCP web site [CNTL-PLANE]
- **Defining domain-specific operations such as path calculation and scheduling algorithms.**

## 1.2 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

When describing abstract data models, this specification uses the notational convention used by the XML Infoset [XML-Infoset]. Specifically, abstract property names always appear in square brackets (e.g., [some property]).

When describing concrete XML schemas, this specification uses a convention where each member of an element's [children] or [attributes] property is described using an XPath-like [XPATH] notation (e.g., /x:MyHeader/x:SomeProperty/@value1).

## 1.3 Terminology

Defined below are the basic definitions for the terminology used in this specification.

**Circuit –** A connection between two endpoints that can be used to transmit data between them.

**Confirmed Inter-domain Path (CIDP)** – A Strict Inter-domain Path (SIDP) where each domain in the path has authorized the use of the path segment between its local ingress and egress links for a specified period of time.

**Control Plane –** The networking infrastructure that is used to share information between entities capable of configuring and managing network equipment. The control plane manages the data plane.

**Data Plane –** Network infrastructure that is used to make data connections between network entities. Devices in the data plane generally correspond to layers 1-3 of the OSI Networking Model [OSI]. A data plane may be managed by a control plane.

**Destination –** The endpoint of a circuit that is the last dynamically controlled link as determined by the direction of the signaling flow.

**Dynamic Circuit Network (DCN) –** A network with a control plane capable of accepting request messages for network resources between two endpoints and provisioning connections based on that request. For the purposes of this document a DCN MUST have a Domain Controller and MAY have an Inter-domain Controller.

**Domain –** In the Network Management Working Group (NMWG) topology schema a set of network devices administrated by a common organization, group, or some other type of authority.

**Domain Controller (DC) –** In the IDC architecture a service that provisions and manages network devices in the local domain.

**Egress -** The property of being a point of exit. The term may be applied to a domain, node, port, or link. When applied to the latter three terms it means the last node/port/link in a given domain. When applied to domain it means the last domain in a given path. An egress node/port/link is equivalent to the destination node/port/link if it is also in the egress domain.

**Endpoint –** The termination points of a dynamic circuit's path. There are two endpoints in a path: source and destination.

**End User –** An entity that sends a request to an Inter-domain Controller (IDC) and is not itself an IDC. The entity may be a human or some type of automated agent.

**Global Reservation Identifier (GRI) -** A name assigned to a reservation upon receiving a reservation creation request. This name is included in all messages about this reservation, including messages about success of the reservation and creation of a circuit from the reservation. The GRI is unique across all domains and is formed by appending a locally unique number to the globally unique domain identifier of the IDC receiving the user request.

**Hop –** An element in a given path. A hop may take the form of a domain, node, port or link.

**Ingress –** The property of being a point of entrance. The term may be applied to a

domain, node, port, or link. When applied to the latter three terms it means the first node/port/link in a given domain. When applied to domain it means the first domain in a given path. An ingress node/port/link is equivalent to the source node/port/link if it is also in the ingress domain.

**Inter-domain Controller (IDC) –** A service that runs in a local domain and coordinates with similar services in other domains to provision network resources across administrative boundaries. Interoperating IDCs create an inter-domain control plane. For the purposes of this document an IDC is a service that implements the IDC protocol.

**Link -** In the NMWG topology schema**,** a connection between two adjacent ports capable of using some subset of resources available on that port.

**Lookup Service –** An external service that maps a human-readable name to a uniform resource name (URN)

**Loose Inter-domain Path (LIDP) –** A list containing two endpoints and zero or more intermediate hops. The hops may take the form of a domain, node, port or link.

**Network Element** – A domain, node, port, or link.

**Network Resource –** A network capability that can be allocated by the control plane. This includes (but is not limited to) bandwidth, VLAN number, and SONET/SDH timeslots.

**Node –** In the NMWG topology schema a physical or logical representation of a junction of ports that connect to other nodes via links. A node may correspond directly to a network device such as a switch or router or may be abstracted to represent a collection of devices such as an Autonomous System (AS).

**Path -** A list of physical or logical network elements in the form of hops that data will traverse when traveling between two endpoints. A path may contain all relevant elements between two endpoints (strict) or only a subset (loose). When a path is instantiated on the network it becomes a circuit.

**Path Segment** – A subset of a path consisting of two or more connected hops.

**Port –** In the NMWG topology schema a physical or logical connection point. A single port may represent one or more interfaces on a network device. Ports are connected by one or more links and are the children of nodes

**Reservation –** The right to use a set of network resources starting at a given time for a

specified duration.

**Signaling –** The process by which Inter-domain Controllers (IDCs) are triggered to have their Domain Controllers (DC) create, manage, and remove circuits associated with a reservation.

**Source –** The endpoint of a circuit that is the first dynamically controlled link as determined by the direction of the signaling flow.

**Strict Inter-domain path (SIDP) –** A list of hops that MUST include every domain's ingress and egress link between its two endpoints. An IDC MUST honor the ingress and egress links specified in the SIDP. A SIDP MAY contain intra-domain hops between a domain's ingress and egress, but there is no requirement to do so. Intra-domain hops MAY be treated as hints in interdomain paths.

In the future, paths may be defined that contain a mixture of strict and loose hops where a strict hop must be honored by the IDC and a loose hop is a hint to the IDC attempting to find a path between endpoints.

**Token –** A hard to counterfeit sequence of bytes that grants the right of the holder to signal a reservation.

**Topology –** A physical or logical description of how devices on the network data plane connect. Elements in the topology may be provisioned by the control plane to create circuits in response to dynamic network resource requests.

**Uniform Resource Name (URN) –** A persistent, location-independent, resource identifier as defined in RFC 2141 [RFC2141]. URNs are used to identify domains, nodes, ports and links in the NMWG topology schema. URNs that reference elements defined in the NMWG topology schema always begin with the prefix "urn:ogf:network". A URN is considered a *fully-qualified identifier* because all parent elements must be defined when referencing elements below the top level of a hierarchical structure. URNs for each element in the domain,-> node -> port ->link hierarchy defined by NMWG look like the following:

- Domain URN: urg:ogf:network:domain=*domain_id*
- Node URN: urg:ogf:network:domain=*domain_id*:node=*node_id*
- Port URN: urg:ogf:network:domain=*domain_id*:node=*node_id*:port=*port_id*
- Link URN:

urg:ogf:network:domain=*domain_id*:node=*node_id*:port=*port_id*:link=*link_id*

## 1.4 Namespaces

The following namespaces are used in this document:

| Prefix | Namespace |
|---|---|
| idc | http://oscars.es.net/OSCARS<br>*See www.controlplane.net for formal specification* |
| nmwg-cp | http://ogf.org/schema/network/topology/ctrlPlane/20070626/<br>*See www.controlplane.net for formal specification* |
| wsse | http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd |
| wsse11 | http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd |
| wsu | http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd |
| wsnt | http://docs.oasis-open.org/wsn/b-2 |
| wsnt-br | http://docs.oasis-open.org/wsn/br-2 |
| ds | http://www.w3.org/2000/09/xmldsig# |
| soap | http://schemas.xmlsoap.org/wsdl/soap12/ |
| xsd | http://www.w3.org/2001/XMLSchema |
| wsdl | http://schemas.xmlsoap.org/wsdl/ |

# 2 Messaging Model

The IDC protocol utilizes a publish/subscribe model for asynchronous messaging. This model is primarily implemented using a message-sequencing scheme described as the “daisy-chain”. This model is described in the section that follows.

*(NOTE: For a discussion of an alternative messaging scheme please see* Appendix B: The Meta-scheduler Model*)*

## 2.1 Publish/Subscribe Model

The Inter-domain Controller Protocol (IDCP) implements a publish/subscribe model as defined by the WS-Notification [WSN] specification from OASIS. Under this model an Inter-domain Controller (IDC) *publishes* events when certain tasks are performed or a failure occurs. Tasks that trigger events include (but are not limited to) the reservation of resources or the creation of a circuit on the network. External IDCs, end-users, or other interested services *subscribe* to receive notification of these events as they are published. External IDCs in particular use the notifications to make decisions about actions to take and when to change a reservation's state.

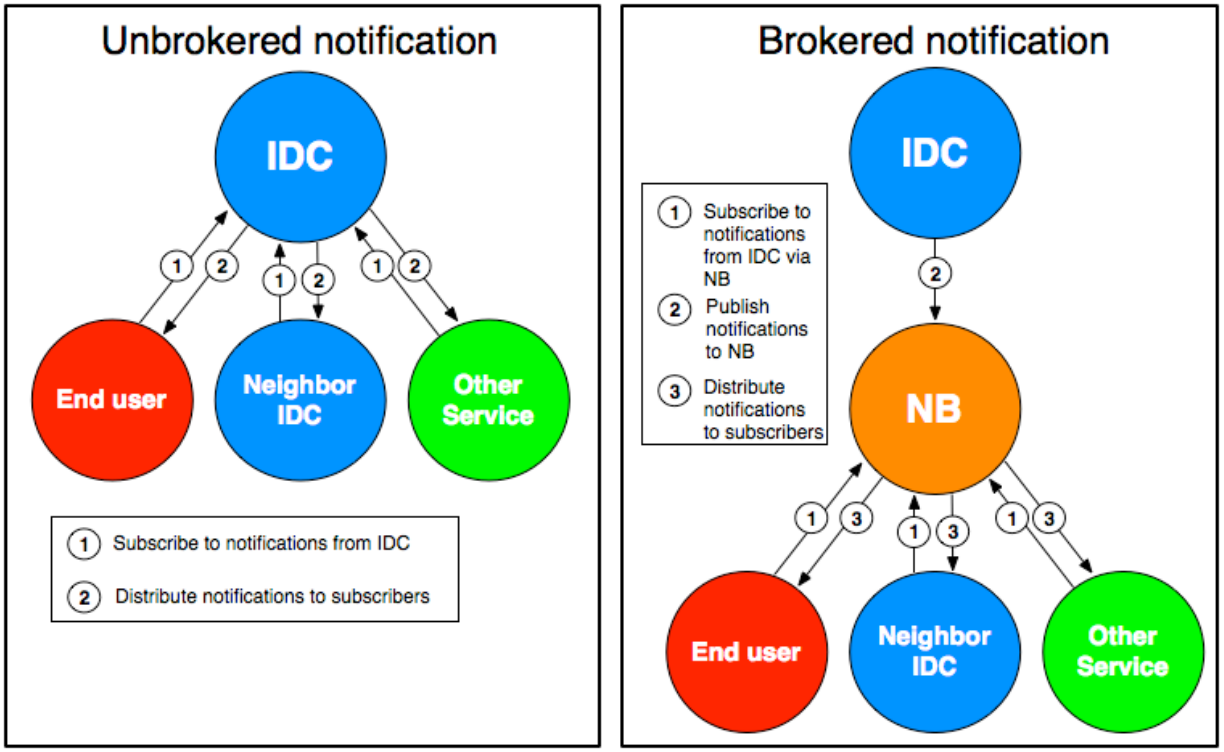


***Figure 2.1*** *(Left) Unbrokered notification where burden of subscription management falls on the IDC. (Right) Brokered notification where subscription management and notification distribution delegated to NotificationBroker (NB)*

The management of subscriptions and distribution of messages can be handled within the IDC service or it can be delegated to an external service using the WS-BrokeredNotification specification [WSBN]. Both methods are shown in *Figure 2.1*. The brokered method uses a service called the NotificationBroker to accept subscriptions and send notifications. The advantage of using the brokered method is that it decreases the logic required in the IDC. The IDC only needs to send a single notification to the NotificationBroker and it will handle the matching and distribution of notifications to

subscribers. The NotificationBroker may also have generally utility for tasks such as network monitoring and topology distribution but those topics are outside the scope of this document and/or reserved for future work.

Subscribers may choose the events for which they receive notification by using a number of parameters. The *topic* is the primary parameter subscribers use to filter notifications. A topic is a pre-defined set of events. The IDC protocol defines the following topics (See **Appendix A** for a full listing of events in each topic**)**:

| Topic Name | Description |
|---|---|
| IDC | Events required between IDCs organized in a daisy-chain (see section 2.2) to reserve resources and provision circuits |
| INFO | Events that contain information about when errors and operations complete but exclude some of the intermediate events of the IDC topic. Useful for end-users or other services that need basic information about IDC activity |
| DEBUG | Events useful for debugging purposes |
| ERROR | This topic contains only those events that indicate an error or failure during an IDC operation. |

Other filters are available that further limit notifications received based on the content of the message. These are described in the corresponding messaging sections of this document.

## 2.2 The Daisy-Chain

The daisy-chain scheme works by passing IDC protocol messages from one IDC to another in a chain-like fashion through a sequence of domains. The order of IDCs in the chain is determined by the path (or expected path) associated with a request. Paths represent a linear sequence of network elements describing how data will travel from one end of a point-to-point circuit to the other. Calculation of the path may be part of the operation being performed (as is the case when a reservation is being created) or may have been calculated by some previous operation (as is the case when cancelling a pre-existing reservation). Since each network element in the linear sequence belongs to an administrative domain an IDC can extrapolate the sequence of domains from the path. The way that messages are passed varies slightly depending on whether the operation is a resource scheduling or signaling request. The sections that follow describe each of these cases.

### 2.2.1 Resource Scheduling Chain

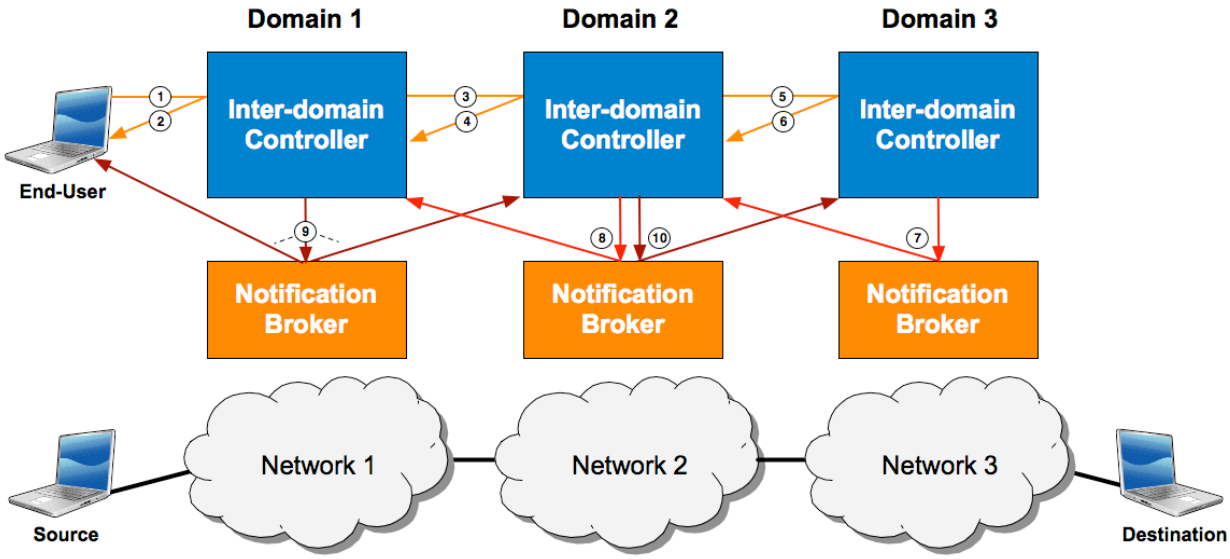


***Figure 2.2*** *An example of a daisy-chain model under the brokered model used during resource scheduling with three domains.*

Resource scheduling operations are those that relate to the creation, modification and cancellation of reservations for network resources. *Figure 2.1* shows an example of a resource scheduling daisy chain between three domains. In the example the brokered notification model is implemented. Details of the various resource operations is saved for a later section but an outline of the basic message-passing process shown in the diagram is described below:

1. The daisy chain is initiated when an end-user sends a request to the IDC of Domain 1, the first domain in the path from source to destination. Currently the end-user MUST send the initial request to the IDC of the first domain in the path.

2. The first IDC does some basic sanity checking of the message and returns an acknowledgement indicating the request was accepted. Further operations are performed asynchronously.

3. Next, the IDC performs some operation depending on the request type, modifies the request as needed, and forwards it to Domain 2

4. Domain 2 responds indicating the request was accepted after checking the request parameters are valid and continues asynchronously.

5. Domain 2 analyzes the request, performs any necessary operations and forwards the message to Domain 3.

6. Domain 3 responds indicating the request was accepted after checking the request parameters are valid and continues asynchronously.
7. Domain 3 is the last domain the path so there is no further forwarding required. Instead it publishes a CONFIRMED event indicating that the operation is finished in the local domain.
8. Domain 2 receives the CONFIRMED event and does some finalization tasks. It then responds with a CONFIRMED message to Domain 1.
9. Domain 1 receives the CONFIRMED event and does some finalization tasks. Since it is the first domain in the path that means every domain has completed their portion of the operation. Domain 1 publishes a COMPLETED event to indicate this fact.
10. Domain 2 passes COMPLETED message to Domain 3. This is purely for informational purposes so that Domain 3 knows the other domains succeeded and so it has any additional information added to the request by the other domains during the confirmation phase.

### 2.2.2 Signaling Chain

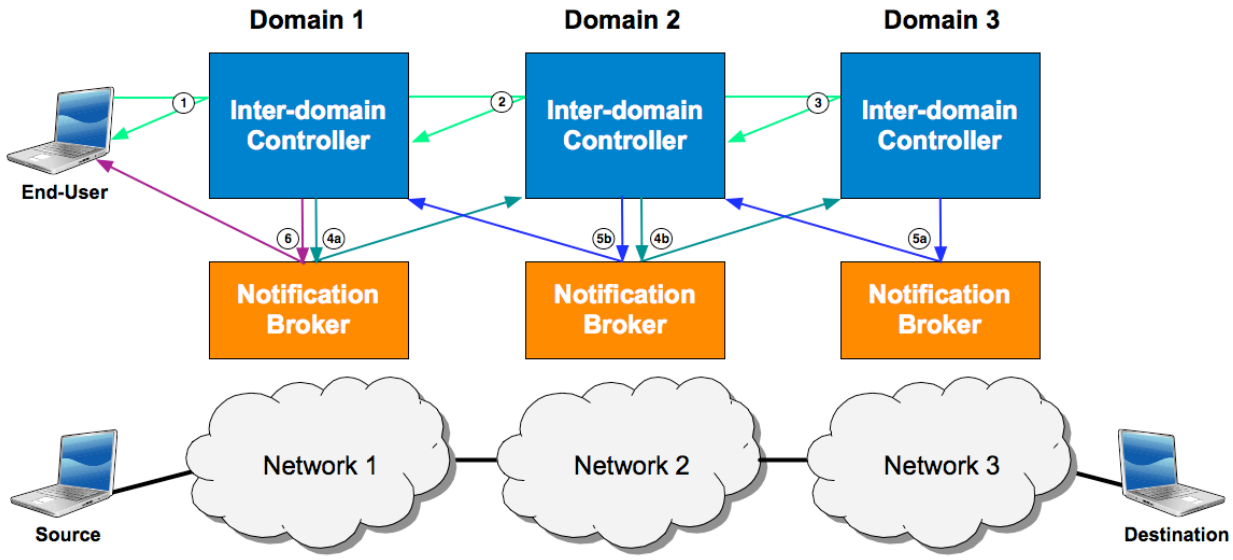


***Figure 2.3*** *Example of how IDCs interact for signaling operations*

Signaling operations are those that trigger the IDC to interact with the domain controller to create and teardown circuits. Figure 2.2 shows the process by which messages are passed for signaling operations. They are described as follows:

1-3. These steps are optional. A circuit may be initiated by sending messages to trigger the setup/ teardown process. Alternatively, the circuit may be triggered in each domain at a start time specified during resource scheduling in which case these steps are skipped. If messages are used they are forwarded down the chain to trigger the process that follows.

4. Each domain sets-up/tears-down their segment of the path in parallel. Two types of events are thrown as each event completes. The first is an UPSTREAM CONFIRMED event. UPSTREAM CONFIRMED events are thrown when every domain *before* the domain publishing the event AND the local domain are finished with their operation. Domains in the example publish the events as follows:

a. Domain 1 publishes this event when it finishes the local setup because it's the first domain in the path.

b. Domain 2 throws the event when the local operation AND domain 1 has published an UPSTREAM CONFIRMED event.

Domain 3 publishes this event when Domain 2 publishes an UPSTREAM CONFIRMED event AND it completes with the local operation. This is not shown in the diagram as the event is also completed at this point.

5. The second event type is a DOWNSTREAM CONFIRMED event. DOWNSTREAM CONFIRMED events are thrown when every domain *after* the domain publishing the event AND the local domain are finished with their operation. Domains in the example publish the events as follows:

a. Domain 3 publishes this event when it finishes the local setup because it's the last domain in the path.

b. Domain 2 throws the event when the local operation completes AND Domain 1 has published a DOWNSTREAM CONFIRMED event.

Domain 1 publishes this event when Domain 2 publishes a DOWNSTREAM CONFIRMED event AND it completes with the local operation. This is not shown in the diagram as the event is also completed at this point.

6. When a domain has published both an UPSTREAM CONFIRMED event and a DOWNSTREAM CONFIRMED event then the operation is complete in every domain. It publishes a COMPLETED event to indicate this. It is published by every domain but only shown in the diagram going to end-user. The other domains do NOT need to wait for a COMPLETED event from the other domains before considering an operation finished.

### 2.2.3 Error Handling Chain

At any point a failure may occur during an operations. When that occurs a FAILED event is published by the IDC where the error originates. It is the responsibility of the neighboring IDCs in the chain to pass the FAILED event to their direct neighbor that didn't experience the error. Likewise, the neighbor must pass the FAILED event farther down the chain. The following scenarios are possible:

- **A failure occurs on the first domain**. When this happens it should pass the failure event to the second domain in the signaling path. The second domain will pass it to the third, etc until the end is reached.
- **A failure occurs in the last domain.** When this happens the last domain should pass the failure event to the second-to-last domain in the signaling path. This domain should pass it to the previous domain before it in the signaling path, etc until the first domain is reached.
- **A failure occurs in the middle domain.** When this happens the domain should pass the failure to its neighbors in both directions. Each neighbor will pass the event along their respective segments of the chain until both ends are reached.

# 3 Reservation States

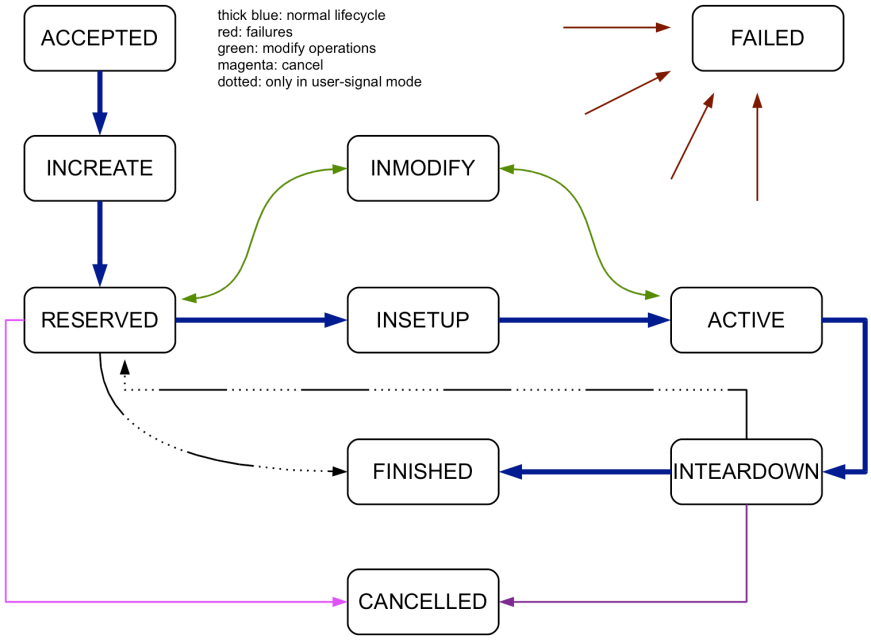


***Figure 3.1*** *State diagram of reservations in the IDC protocol*

The IDC protocol creates resource reservations that change state in response to protocol messages. *Figure 3.1* shows the states and transitions for reservations created using the IDC protocol. The following describes each state:

- **ACCEPTED** – A reservation is in this state when a user first submits a request to create a reservation and its parameters have been validated but prior to performing an initial path computation.
- **INCREATE** – A reservation is in this state when path computation begins and until the RESERVATION_CREATE_COMPLETED or RESERVATION_CREATE_FAILED is triggered. In other words, a reservation is in this state until every domain has finished attempting to reserve the requested resources.
- **PENDING** – A reservation in this state has resources reserved in all domains (as signaled by a RESERVATION_CREATE_COMPLETED event) but the start time of the underlying resource has not been reached. In addition, if the reservation uses XML signaling it will be in this state until a createPath message is received. A reservation may also be cancelled while in this state.
- I**NSETUP** – A reservation is in this state while the circuit is being created on the network. A circuit remains in this state until a PATH_SETUP_COMPLETED event or PATH_SETUP_FAILED event occurs. In other words, a circuit remains in this state until every domain in the path has completed the circuit creation process (either successfully or unsuccessfully).
- **ACTIVE** - A reservation enters this state when every domain has completed creating the circuit as signaled by a PATH_SETUP_COMPLETED event. A circuit remains in this state until the reservation end time or a user action (such as a modification) changes the state.
- **INMODIFY** – A reservation is in this state if it was previously PENDING or ACTIVE but a user has requested it be modified. A reservation will return to its previous PENDING or ACTIVE state once the modification completes (whether successfully or unsuccessfully).
- **INTEARDOWN** – This event occurs while domains are removing a circuit. A reservation MUST enter this state when the reservation end time is reached. A failure, user-request teardown, or cancellation may also change the reservation to this state.

- **CANCELLED** – A reservation enters this state after all domains in the path have finished cancelling a reservation as signaled by a RESERVATION_CANCEL_COMPLETED message. When a reservation is cancelled the circuit is removed from the network if cancellation occurs while circuit is in the ACTIVE state. Also, the hold on resources is released so they are free for reservation. This is a terminal state meaning the reservation state cannot change once in CANCELLED.
- **FINISHED** – A reservation enters this state when the end time is reached AND every domain in the path has removed their circuit from the network (as defined by the RESERVATION_TEARDOWN_COMPLETED event). This is a terminal state meaning the reservation state cannot change once in FINISHED.
- **FAILED –** A reservation may reach this state due to a non-recoverable error in any of the above states occurs. The state is reached if any of the FAILED events detailed in Appendix A: IDC Topics and Events occur. This is a terminal state meaning the reservation state cannot change once in FAILED.

# 4 Security

## 4.1 Authentication and Authorization

The IDC uses SOAP messages, secured by WS-Security v1.1 [WS-Sec] using the XML Signature standard [DigSig] to timestamp and sign, but not encrypt the message body for the request messages and to timestamp, but not sign or encrypt the reply messages. The messages are SOAP over HTTPS with server-side authentication that serves to authenticate the HTTPS server to the client and to encrypt the connection. The message signature accomplishes end-to-end authentication of the requester to the IDC server. Note that at some sites the HTTPS server is on a separate host from the IDC server due to firewall constraints. The IDC expects to find the x.509 certificate of the requester included in the digital signature. It verifies that certificate and extracts the subject name from the certificate which it uses to authorize the requested action. Note that in order to verify the included certificate the IDC must have access to a trusted copy of the certificate of its issuer. The privileges of a given requester are kept locally by the IDC and indexed by the user's subject name. They are not currently part of the message protocol. If in the future it is desired to identify users by some means other than an x.509 certificate, for example a Kerberos token or a SAML assertion, the IDC will need to be modified to use such an identifier to access the privilege information for the user.

## 4.2 Digital Signature Format and Algorithms

In theort, the IDC protocol should support a variety of algorithms used by the digital signature. The only part of the signature that the IDC depends on is the security token being an x.509 certificate from which the name of the requester can be extracted. To the extent that various XML signing libraries support different algorithms one should be able to choose various canonicalization, transforms, digest and signature methods.

However, due to the lack of compatibility of different packages and languages, it is strongly recommended to use the choices shown in the example and itemized below:

Signing Info: the entire body of the message is signed in one part.

KeyInfo: the security token is a base64 encoded binary x.509v3 certificate.

Canonicalization method: Exclusive XML canonicalization, (http://www.w3.org/2001/10/xml-exc-c14n#), is strongly recommended by the WS-security specification. See [WS-Sec] section 8.1.

Transform method: same as canonicalization method

Digest method: SHA1 (http://www.w3.org/2000/09/xmldsig#sha1) is considered more secure than md5 the other widely used digest algorithm.

Signature method: rsa-sha1 (http://www.w3.org/2000/09/xmldsig#rsa-sha1) is the standard method to use an rsa key to sign a sha1 digest of the text.

## 4.3 Example

### 4.3.1 Request message

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
       200401-wss-wssecurity-secext-1.0.xsd"
      soap:mustUnderstand="true">
      <wsse:BinarySecurityToken
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
          200401-wss-wssecurity-utility-1.0.xsd"
        EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-
          1.0#Base64Binary"
        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
```

```
          200401-wss-x509-token-profile-1.0#X509v3"
        wsu:Id="CertId-6479960">
        [X.509 Certificate]
      </wsse:BinarySecurityToken>
      <ds:Signature
         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
         Id="Signature-1830472">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-
              c14n#"/>
          <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
              sha1"/>
          <ds:Reference URI="#id-7438423">
            <ds:Transforms>
              <ds:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-
                c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod
               Algorithm="http://www.w3.org/2000/09/
               xmldsig#sha1"/>
            <ds:DigestValue>
              [SHA 1 Digest]
            </ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>
           [Message Signature]
        </ds:SignatureValue>
        <ds:KeyInfo Id="KeyId-15565667">
          <wsse:SecurityTokenReference
            xmlns:wsu="http://docs.oasis-
              open.org/wss/2004/01/oasis-200401-wss-wssecurity-
              utility-1.0.xsd"
            wsu:Id="STRId-13122813">
            <wsse:Reference URI="#CertId-6479960"
              ValueType="http://docs.oasis-
                open.org/wss/2004/01/oasis-200401-wss-x509-
                token-profile-1.0#X509v3"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
      <wsu:Timestamp
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
```

```
          200401-wss-wssecurity-utility-1.0.xsd"
        wsu:Id="Timestamp-13182325">
        <wsu:Created>2008-05-05T19:43:25.596Z</wsu:Created>
        <wsu:Expires>2008-05-05T19:48:25.596Z</wsu:Expires>
      </wsu:Timestamp>
    </wsse:Security>
  </soap:Header>
  <soap:Body
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
      200401-wss-wssecurity-utility-1.0.xsd"
    wsu:Id="id-7438423">
    [Unencrypted IDC Message]
  </soap:Body>
</soap:Envelope>
```

### 4.3.2 Reply message

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
        200401-wss-wssecurity-secext-1.0.xsd"
      soap:mustUnderstand="true">
      <wsu:Timestamp
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
          200401-wss-wssecurity-utility-1.0.xsd"
        wsu:Id="Timestamp-5830260">
        <wsu:Created>2008-05-05T19:43:32.635Z</wsu:Created>
        <wsu:Expires>2008-05-05T19:48:32.635Z</wsu:Expires>
      </wsu:Timestamp>
      <wsse11:SignatureConfirmation
        xmlns:wsse11="http://docs.oasis-open.org/wss/oasis-wss-
        wssecurity-secext-1.1.xsd"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
          200401-wss-wssecurity-utility-1.0.xsd"
        Value="[Signature Value]"
        wsu:Id="SigConf-707092" />
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    [Unencrypted IDC Message Response]
  </soap:Body>
</soap:Envelope>
```

# 5 End-User to IDC Interface

The IDC protocol defines a SOAP [SOAP] interface between the end-user and IDC that MAY be implemented by a particular IDC instance. An IDC instance MAY implement (instead or in addition) a custom interface for end-user interaction and still be valid if the messages passed between IDCs conform to this specification document. An IDC SHOULD implement some type of end-user interface that allows requesters to initiate the operations defined in this specification

The end-user interface defined by this specification uses SOAP messages similar to those passed between IDCs. The SOAP header contains the elements defined by WS-Security [WS-Sec] and described in section **4** of this document. The SOAP body of messages may be one of the several types defined in sections **9** thru **10**. The primary difference between the body of messages exchanged between an end-user and those exchanged with another IDC is that the former are not encapsulated in <idc:forward> or <idc:forwardResponse> elements (see section **6**).

The WSDL [WSDL] operations available for end-user interactions with the IDC as defined by this interface are listed below:

```
<wsdl:operation name="createReservation">
    <wsdl:input message="tns:createReservation" />
    <wsdl:output message="tns:createReservationResponse" />
    <wsdl:fault name="AAAErrorException"
        message="tns:AAAFaultMessage" />
    <wsdl:fault name="BSSErrorException"
        message="tns:BSSFaultMessage" />
</wsdl:operation>
<wsdl:operation name="cancelReservation">
    <wsdl:input message="tns:cancelReservation"></wsdl:input>
    <wsdl:output message="tns:cancelReservationResponse" />
    <wsdl:fault name="AAAErrorException"
        message="tns:AAAFaultMessage" />
    <wsdl:fault name="BSSErrorException"
        message="tns:BSSFaultMessage" />
</wsdl:operation>
<wsdl:operation name="queryReservation">
    <wsdl:input message="tns:queryReservation" />
    <wsdl:output message="tns:queryReservationResponse" />
    <wsdl:fault name="AAAErrorException"
        message="tns:AAAFaultMessage" />
```

```
    <wsdl:fault name="BSSErrorException"
        message="tns:BSSFaultMessage" />
</wsdl:operation>
<wsdl:operation name="modifyReservation">
    <wsdl:input message="tns:modifyReservation" />
    <wsdl:output message="tns:modifyReservationResponse" />
    <wsdl:fault name="AAAErrorException"
        message="tns:AAAFaultMessage" />
    <wsdl:fault name="BSSErrorException"
        message="tns:BSSFaultMessage" />
</wsdl:operation>

<wsdl:operation name="listReservations">
    <wsdl:input message="tns:listReservations" />
    <wsdl:output message="tns:listReservationsResponse" />
    <wsdl:fault name="AAAErrorException"
        message="tns:AAAFaultMessage" />
    <wsdl:fault name="BSSErrorException"
        message="tns:BSSFaultMessage" />
</wsdl:operation>
<wsdl:operation name="createPath">
    <wsdl:input message="tns:createPath" />
    <wsdl:output message="tns:createPathResponse" />
    <wsdl:fault name="AAAErrorException"
        message="tns:AAAFaultMessage" />
    <wsdl:fault name="BSSErrorException"
        message="tns:BSSFaultMessage" />
</wsdl:operation>
<wsdl:operation name="teardownPath">
    <wsdl:input message="tns:teardownPath" />
    <wsdl:output message="tns:teardownPathResponse" />
    <wsdl:fault name="AAAErrorException"
        message="tns:AAAFaultMessage" />
    <wsdl:fault name="BSSErrorException"
        message="tns:BSSFaultMessage" />
</wsdl:operation>
```

A detailed description of each message type in the context of end-user requests as well as IDC-to-IDC requests is described in sections **9** thru **12** of this document.

# 6 IDC to IDC Message Forwarding

The initial messages passed between IDCs along a daisy chain use the *forward* operation. These are only for the initial requests that start an operation, not the notification messages to confirm and complete an operation (see the Notify section). The WSDL [WSDL] definition of the *forward* operation is shown below:

```
<wsdl:operation name="forward">
    <wsdl:input message="tns:forward"></wsdl:input>
    <wsdl:output message="tns:forwardResponse"></wsdl:output>
    <wsdl:fault name="AAAErrorException"
        message="tns:AAAFaultMessage" />
    <wsdl:fault name="BSSErrorException"
        message="tns:BSSFaultMessage" />
</wsdl:operation>
```

The operation defines an <idc:forward> element included in the SOAP body of a message. The XML Schema [XML Schema] definition for this element is described below:

```
<xsd:element name="forward">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="payload" type="tns:forwardPayload" />
      <xsd:element name="payloadSender" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/forward

Container element included in the body of SOAP message when an IDC is sending a request to the next IDC in a daisy chain

/forward/payload

Contains message element with request-specific parameters

/forward/payloadSender

A string indicating the end-user that sent the initial request. This string may be a domain specific value such as a login associated with the end-user. Future versions of this specification may further define this element.

A forward request will contain a different payload depending on the operation being performed. Below is an XML Schema [XML Schema] description of the payload type:

```
<xsd:complexType name="forwardPayload">
  <xsd:sequence>
    <xsd:element name="contentType" type="xsd:string" />
    [Message Content Element]
  </xsd:sequence>
</xsd:complexType>
```

/forward/contentType

A string value corresponding to the element name of [Message Content Element] in this request

/forward /[Message Content Element]

The message being forwarded as indicated by /forward/contentType. Valid request types are those indicated in sections **9** and **10.**

The *forward* operation further defines a <idc:forwardResponse> message to be returned when an IDC is done processing a <idc:forward> request. The <idc:forwardResponse> element and its type are defined below:

```
<xsd:element name="forwardResponse" type="tns:forwardReply" />
<xsd:complexType name="forwardReply">
  <xsd:sequence>
    <xsd:element name="contentType" type="xsd:string" />
    [Message Content Element]
  </xsd:sequence>
</xsd:complexType>
```

/forwardResponse

A container element included in the SOAP body of a message responding to an earlier <idc:forward> request

/fowardResponse/contentType

String value corresponding to the element name of the [Message Content Element] content in this response

/fowardResponse /[Message Content Element]

The response element indicated by /fowardResponse/contentType and corresponding to the orginal <idc:forward> request. Valid responses are those indicated in sections **9** and **10** of this document.

# 7 Common Data Types

Data types common to several messages are described in this section.

## 7.1 Reservation Details

All reservations are described by the following XML Schema definition. All elements in this definition MUST be included.

```
<xsd:complexType name="resDetails">
  <xsd:sequence>
    <xsd:element name="globalReservationId" type="xsd:string" />
    <xsd:element name="login" type="xsd:string" />
    <xsd:element name="status" type="xsd:string" />
    <xsd:element name="startTime" type="xsd:long" />
     <xsd:element name="endTime" type="xsd:long" />
     <xsd:element name="createTime" type="xsd:long" />
     <xsd:element name="bandwidth" type="xsd:int" />
     <xsd:element name="description" type="xsd:string" />
     <xsd:element name="pathInfo" type="tns:pathInfo" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:ResDetails;/idc:globalReservationId

The unique GRI described in section 1.3.

/idc:ResDetails/idc:login

The login identifier for the user on the originating IDC.

/idc:ResDetails/idc:status

Contains the current reservation status. See section **3** for a list of valid values.

/idc:ResDetails/idc:startTime

Contains the time the circuit was set up, if it was set up successfully. It is in seconds since the epoch (Unix time).

/idc:ResDetails/idc:endTime

Contains the time the circuit is to be torn down or was torn down. It is in seconds since the epoch (Unix time).

/idc:ResDetails/idc:createTime

Contains the time the reservation was scheduled, if it was scheduled successfully. It is in seconds since the epoch (Unix time).

/idc:ResDetails/idc:bandwidth

The bandwidth for the circuit in megabits per second (Mbps).

/idc:ResDetails/idc:description

Contains a human-readable description of the reservation's purpose.

/idc:ResDetails/idc:pathInfo

The next section describes all the definitions involved in the path involved with a circuit.

## 7.2 Path Information

```
<xsd:complexType name="pathInfo">
  <xsd:sequence>
    <xsd:element name="pathSetupMode" type="xsd:string"
      minOccurs="1" />
    <xsd:element name="pathType" type="xsd:string" maxOccurs="1"
      minOccurs="0" />
    <xsd:element name="path" type="nmwg-cp:CtrlPlanePathContent"
      maxOccurs="1" minOccurs="0" />
    <xsd:element name="layer2Info" type="tns:layer2Info"
      maxOccurs="1" minOccurs="0" />
    <xsd:element name="layer3Info" type="tns:layer3Info"
      maxOccurs="1" minOccurs="0" />
    <xsd:element name="mplsInfo" type="tns:mplsInfo"
      maxOccurs="1" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
```

idc:pathInfo/idc:pathSetupMode

This field MUST be included and is an indicator to the scheduler whether it should initiate circuit setup automatically (see section **10.1**) or have the user initiate circuit setup with the *createPath* message (see section **10.2.1)**.

idc:pathInfo/idc:pathType

This field MAY be included, and indicates whether a path is "strict" or "loose". If not included then the path is assumed to be "strict". A "strict" path indicates that path is a Strict Inter-domain Path (SIDP) which (by definition) means that the circuit MUST be setup using the specified ingress and egress points exactly as given. A value of "loose" indicates that this is a Loose Inter-domain Path (LIDP) and that IDCs may expand and modify segments of the path during reservation scheduling.

idc:pathInfo/idc:path

This element contains the current set of hops in a given path. The contents of this element are defined in the NMWG Control Plane topology schema [NMWG-CP]. If idc:pathInfo/idc:pathType is set to “loose” then the hops inside this element may be domain, node, port of link URNs. If idc:pathInfo/idc:pathType is not included or set to strict then they MUST be link URNs

The following is excerpted from the NMWG topology schema:

```
<xs:complexType name="CtrlPlanePathContent">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="hop"
      type="CtrlPlane:CtrlPlaneHopContent" />
  </xs:sequence>
  <xs:attribute name="id" use="required" type="xs:string"/>
</xs:complexType>
```

nmwg-cp:CtrlPlanePathContent

Contains a series of hops along the associated path.

nmwg-cp:CtrlPlanePathContent/nmwg:id

Contains the id of the associated path.

```
<xs:complexType name="CtrlPlaneHopContent">
  <xs:sequence>
    <xs:element minOccurs="0" name="domainIdRef"
      type="xs:string" />
    <xs:element minOccurs="0" name="nodeIdRef"
      type="xs:string" />
    <xs:element minOccurs="0" name="portIdRef"
      type="xs:string" />
    <xs:element minOccurs="0" name="linkIdRef"
      type="xs:string" />
    <xs:element minOccurs="0" ref="CtrlPlane:domain"/>
    <xs:element minOccurs="0" ref="CtrlPlane:node"/>
    <xs:element minOccurs="0" ref="CtrlPlane:port"/>
    <xs:element minOccurs="0" ref="CtrlPlane:link"/>
    <xs:element minOccurs="0" maxOccurs="unbounded"
      name="nextHop" type="CtrlPlane:CtrlPlaneNextHopContent" />
  </xs:sequence>
  <xs:attribute name="id" use="required" type="xs:string"/>
  </xs:complexType>
```

A hop contains an optional link, port, node, and domain id, and a hop id in the NMWG URN format. It should be noted that prior to a reservation reaching the PENDING state that any of the values under hop are valid. After a reservation becomes PENDING, though, it is expected that hops will contain full link elements. Hops also contain a nextHop element that's value is the ID of the next hop in the path. It can also be weighted and listed as optional during resource scheduling if its not required to be used in the final path.

One of the <idc:layer2Info> or <idc:layer3Info> types MUST be present. These types contain information dependent on whether the underlying technology of the path to be set up operates at OSI layer 2 or layer 3. The <idc:mplsInfo> type MAY be present, depending on whether the MPLS protocol is being used in the particular IDC.

The following describes the layer2Info type:

```
<xsd:complexType name="layer2Info">
  <xsd:sequence>
    <xsd:element name="srcVtag" type="tns:vlanTag" minOccurs="0"
      maxOccurs="1" />
    <xsd:element name="destVtag" type="tns:vlanTag"
      minOccurs="0" maxOccurs="1" />
    <xsd:element name="srcEndpoint" type="xsd:string" />
    <xsd:element name="destEndpoint" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="vlanTag">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute use="optional" name="tagged"
        type="xsd:boolean"/>
      </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

idc:vlanTag

Contains a string for the VLAN id and a boolean which MAY be included indicating whether this VLAN is tagged or not.

idc:layer2Info/idc:srcVtag

This field MAY be included and specifies the VLAN at the source and whether it is tagged or not.

idc:layer2Info/idc:destVtag

This field MAY be included and specifies the VLAN at the destination and whether it is tagged or not.

idc:layer2Info/idc:srcEndpoint

This field MUST be included, and contains an identifier for the source at the ingress of the originating IDC.

idc:layer2Info/idc:destEndpoint

This field MUST be included, and contains an identifier for the destination at the egress of the ending IDC.

The layer3Info type is as follows:

```
<xsd:complexType name="layer3Info">
  <xsd:sequence>
    <xsd:element name="srcHost" type="xsd:string" />
    <xsd:element name="destHost" type="xsd:string" />
    <xsd:element name="protocol" type="xsd:string"
      maxOccurs="1" minOccurs="0"/>
    <xsd:element name="srcIpPort" type="xsd:int" maxOccurs="1"
      minOccurs="0" />
    <xsd:element name="destIpPort" type="xsd:int" maxOccurs="1"
      minOccurs="0"/>
    <xsd:element name="dscp" type="xsd:string" maxOccurs="1"
      minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
```

idc:layer3Info/idc:srcHost

This field MUST be included, and contains the DNS name or the IP address of the source of the path.

idc:layer3Info/idc:destHost

This field MUST be included, and contains the DNS name or the IP address of the destination of the path. The source and destination are typically outside the scope of a particular IDC, and the path may also contains hops outside the scope of an IDC.

idc:layer3Info/idc:protocol

This field MAY be included, and is typically “udp” or “tcp”, though other protocols may be specified.

idc:layer3Info/idc:srcIpPort

This field MAY be included, and is the transport-layer port number at the source.

idc:layer3Info/idc:destIpPort

This field MAY be included, and is the transport-layer port number at the destination.

idc:layer3Info/idc:dscp

This field MAY be included, and contains the Differentiated Services Code Point used in QoS.

The mplsInfo type MAY be present, and is only used where the IDC uses MPLS:

```
<xsd:complexType name="mplsInfo">
  <xsd:sequence>
    <xsd:element name="burstLimit" type="xsd:int" />
    <xsd:element name="lspClass" type="xsd:string"
      maxOccurs="1" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
```

idc:mplsInfo/idc:burstLimit

This field MUST be present, and is used by the policer to determine the maximum burst above the average bandwidth.

idc:mplsInfo/idc:lspClass

This field MUST be present, and contains the MPLS class of service

## 7.3 Events

Events are wrapped in notifications that indicate when a certain task has completed or a failure has occurred. They are described as follows:

```
<xsd:complexType name="eventContent">
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:sequence>
    <xsd:element name="type" type="xsd:string" />
    <xsd:element name="timestamp" type="xsd:long" />
    <xsd:element name="userLogin" type="xsd:string"
      maxOccurs="1" minOccurs="0" />
    <xsd:element name="errorSource" type="xsd:string"
      maxOccurs="1" minOccurs="0" />
    <xsd:element name="errorCode" type="xsd:string"
      maxOccurs="1" minOccurs="0" />
    <xsd:element name="errorMessage" type="xsd:string"
      maxOccurs="1" minOccurs="0" />
    <xsd:element name="resDetails" type="tns:resDetails"
      maxOccurs="1" minOccurs="0" />
    <xsd:element name="msgDetails" type="tns:msgDetails"
```

```
      maxOccurs="1" minOccurs="0" />
    <xsd:element name="localDetails" type="tns:localDetails"
      maxOccurs="1" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
```

@id

Required. A string unique to the notification producer that can be used to identify the event. No format is specified for this identifier currently by this protocol so any string is valid.

idc:timestamp

Required. The date and time that the event occurred as the number of seconds since the epoch.

idc:type

The type of event. See Appendix A: IDC Topics and Events and the individual messaging sections for valid event types.

idc:userLogin

A string representing the user login that triggered the event (may be different from idc:resDetails/idc:login if user that triggered event is not the user that placed original reservation).

idc:errorSource

A string with the domain ID of the domain where an error originated. This field is not used if the event is not an error event.

idc:errorCode

If an error occurred then this contains a string representing the error code of the event. Currently no error codes are defined by the IDC protocol and this field is considered a placeholder for the future. This field is never used if the event is not an error event.

idc:errorMessage

If an error occurred this element contains further information about the error. This field is not used if the event is not an error event.

idc:resDetails

A list of details about a particular reservation including the time, path, and bandwidth. This element is required for resource scheduling events and optional

for all others. It SHOULD be included in all events so the subscriber is not required to discover reservation details by other means.

idc:msgDetails
: Optional element containing the full XML message that triggered the event. It contains any of the message types specified in this document for resource scheduling, signaling or polling. It SHOULD be used for notifications belonging to the DEBUG topic.

idc:localDetails
: Optional opaque element containing the additional local information about an event. The contents of this event are one or more xsd:any elements. It MAY include things like the full local path of a particular circuit in the domain owning the IDC that produced the event. When used IDCs each party MUST NOT make any assumption about the contents of this field.

# 8 Notification Interface

The notification interface is used to push messages between IDCs and clients listening for certain events from the IDC. This interface is defined in the WS-Notification specification. This section defines the core messages used to distribute notifications. Also see sections **13** Brokered Notification and **14** Advanced Subscription Management for additional call that are not required by the IDC protocol but may be useful in some implementations.

## 8.1 Subscribe

### 8.1.1 Request

A *Subscribe* message as defined by the WS-Notification [WSN] specification is sent by parties that wish to receive notifications about an event to either the NotificationBroker or directly to the IDC (depending on the implementation). An IDC MUST send this to its neighbors if *Notify* messages do not provide an authentication mechanism such as WS-Security or a two-way SSL handshake. In such a case the *Subscribe* request and response can exchange a unique ID that authenticates the sender of the notification to the receiver. A neighboring IDC MAY send a subscribe message if the *Notify* messages contain WS-Security headers but this is not required. A complete description of the Subscribe message can be found in the WS-Notification specification. The format of the Subscribe message in the context of the IDC is provided below:

```
<xsd:element name="Subscribe" >
  <xsd:complexType>
```

```
    <xsd:sequence>
      <xsd:element name="ConsumerReference"
       type="wsa:EndpointReferenceType"
       minOccurs="1" maxOccurs="1" />
      <xsd:element name="Filter"
       type="wsnt:FilterType"
       minOccurs="0" maxOccurs="1" />
      <xsd:element name="InitialTerminationTime"
       type=" wsnt:AbsoluteOrRelativeTimeType" nillable="true"
       minOccurs="0" maxOccurs="1" />
      <xsd:element name="SubscriptionPolicy"
       minOccurs="0" maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:any namespace="##any" processContents="lax"
             minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:Subscribe

A container element for subscription parameters

/wsnt:Subscribe/wsnt:ConsumerReference

A field indicating the URL where *Notify* messages that match this subscription should be sent. This field is an EndpointReference type as defined in WS-Addressing [WSA]. The Address field MUST contain the URL to send notifications. The URL SHOULD point to an endpoint running HTTPS.

/wsnt:Subscribe/wsnt:Filter

This field contains one or more constraints a *Notify* message must meet to match the subscription. This field MUST contain at least one top as defined in WS-Notification [WSN]. It SHOULD also contain the IDC from which it is wishes to see notifications. Additional filters such as those based on XPath MAY be included but are not required by the IDC protocol.

/wsnt:Subscribe/wsnt:InitialTerminationTime

Optional field. This field contains a suggested time that the subscription will expire. An IDC or NotificationBroker MAY reject a subscription if an InitialTerminationTime is too far in the future or it may ignore it.

/wsnt:Subscribe/wsnt:SubscriptionPolicy

Optional field. An opaque type that specifies further policy about the subscription. This field is undefined for the purposes of the IDC protocol but MAY be included if a particular implementation supports this field.

### 8.1.2 Response

After a subscription is received and processed a NotificationBroker (Or IDC depending on the implementation) responds with the following message:

```
<xsd:element name="SubscribeResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="SubscriptionReference"
       type="wsa:EndpointReferenceType"
       minOccurs="1" maxOccurs="1" />
      <xsd:element ref="wsnt:CurrentTime"
       minOccurs="0" maxOccurs="1" />
      <xsd:element ref="wsnt:TerminationTime"
       minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:SubscribeResponse

A container element for the parameters in the response to the subscription

/wsnt:SubscribeResponse/wsnt:SubscriptionReference

A field that identifies the subscription. It is a WS-Addressing EndpointReference type that contains the URL of the service providing the subscription (i.e. the NotificationBroker or IDC) and a unique identifier of the subscription. This pair of values MUST be included in all Notify messages sent to the consumer of this subscription. See **Section 8.1.3** for a detailed description of this field.

/wsnt:SubscribeResponse/wsnt:CurrentTime

Optional field. The time when the response was sent.

/wsnt:SubscribeResponse/wsnt:TerminationTime

Optional field. The time when the subscription expires.

### 8.1.3 Identifying Subscriptions

Subscriptions are identified using the *wsnt:SubscriptionReference* field. As defined in WS-Notification [WSN] the *wsnt:SubscriptionReference* is an WS-Addressing EndpointReferenceType. For the purposes of the IDC this field MUST take a special format beyond what is required in external specification. The the */wsnt:SubscriptionReference/wsa:Addess* field MUST be the URL to the service providing the subscription such as the NotificationBroker or IDC (depending on the implementation). In addition the */wsnt:SubscriptionReference/wsa:ReferenceParameters* field MUST contain an *idc:subscriptionId* field that contains an identifier unique to subscription service. This may be a UUID or some other locally generated field. The combination of the subscription service's URL and the locally unique ID ensure a globally unique subscription. An example SubscriptionReference is shown below:

```
<wsnt:SubscriptionReference>
    <wsa:Address>
       https://mydomain.net/NotificationBroker
    </wsa:Address>
    <wsa:ReferenceParameters>
        <idc:subscriptionId>
            urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
        </idc:subscriptionId>
    </wsa:ReferenceParameters>
</wsnt:SubscriptionReference>
```

### 8.1.4 Example

An example of a *Subscribe* message is shown below:

```
<s:Envelope ... >
  <s:Header>
    <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:Subscribe>
      <wsnt:ConsumerReference>
        <wsa:Address>
           https://domain1.net/IDC
```

```
        </wsa:Address>
      </wsnt:ConsumerReference>
      <wsnt:Filter>
        <wsnt:TopicExpression Dialect=" http://docs.oasis-
          open.org/wsn/t-1/TopicExpression/Full">
          idc:IDC
        </wsnt:TopicExpression>
        <wsnt:ProducerProperties
           Dialect="http://www.w3.org/TR/1999/REC-xpath-
           19991116">
         /wsa:Address='https://domain2.net/IDC'
        </wsnt:ProducerProperties>
      </wsnt:Filter>
    </wsnt:Subscribe>
  </s:Body>
</s:Envelope>
```

In this example Domain 1 is subscribing to message from Domain 2's NotificationBroker. The wsnt:ConsumerReference field indicates Domain 1 wants Notify messages sent to "https://domain1.net/IDC". The *wsnt:Filter* indicates Domain 1 is subscribing to the idc:IDC topic and only want notifications produced by the service at "https://domain2.net/IDC" (i.e. Domain 2's IDC).

After receiving this request Domain 2 responds indicating the Subscription was a success with the following message:

```
<s:Envelope ... >
  <s:Header>
   <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:SubscribeResponse>
      <wsnt:SubscriptionReference>
        <wsa:Address>
           https://domain2.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
          <idc:subscriptionId>
            urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
          </idc:subscriptionId>
        </wsa:ReferenceParameters>
      </wsnt:SubscriptionReference>
      <wsnt:CurrentTime>
        2009-10-01T14:02:30.000+00:00
      </wsnt:CurrentTime>
      <wsnt:TerminationTime>
```

```
        2009-10-01T15:02:30.000+00:00
      </wsnt:TerminationTime>
    </wsnt:SubscribeResponse>
  </s:Body>
</s:Envelope>
```

We can see that the Domain 2 NotificationBroker created a *SubscriptionReference* to identify the subscription. It also provided the current time and indication that the subscription will expire in one hour (if not renewed or cancelled).

## 8.2 Notify

### 8.2.1 Message Format

*Notify* messages are sent between IDCs and to other interested parties when an event occurs. These messages are asynchronous in the sense that no SOAP response is expected from the consumer. Examples of this type of message are provided in sections **9** and **10** when discussing the specific applications of this message. The format of the Notify message is shown below:

```
<xsd:element name="Notify" >
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsnt:NotificationMessage"
       minOccurs="1" maxOccurs="unbounded" />
      <xsd:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:Notify

A container element for one or more messages

/wsnt:Notify/wsnt:NotifcationMessage

An element or type *wsnt:NotificationMessageHolderType* containing the notification that occurred. See the next heading for a detailed description of this type.

A more detailed view of the idc:NotifcationMessage is provided below:

```
<xsd:complexType name="NotificationMessageHolderType" >
  <xsd:sequence>
    <xsd:element ref="wsnt:SubscriptionReference"
     minOccurs="0" maxOccurs="1" />
    <xsd:element ref="wsnt:Topic"
     minOccurs="0" maxOccurs="1" />
```

```
    <xsd:element ref="wsnt:ProducerReference"
     minOccurs="0" maxOccurs="1" />
    <xsd:element name="Message" type="wsnt:MessageType"/>
  </xsd:sequence>
</xsd:complexType>
```

/wsnt:NotifcationMessage/wsnt:SubscriptionReference

A refererence identifying the subscription which caused this notification to be sent to the consumer. It MUST match the identifier provided in the *SubscribeResponse* message.

/wsnt:NotifcationMessage/wsnt:Topic

The topic or list of topics to which this notification belongs.

/wsnt:NotifcationMessage/wsnt:ProducerReference

An endpoint reference identifying the service (i.e. IDC) that generated this notification. If this message is between an IDC and the local NotificationBroker then it MUST contain a *idc:publisherRegistrationId* element in the *wsa:ReferencePameters*. If it is between a NotificationBroker and a consumer it MUST NOT contain the *idc:publisherRegistrationId*.

/wsnt:NotifcationMessage/wsnt:Message

In the IDC protocol this element MUST contain at least one *idc:event* element as defined in section **7.3** Events.

## 8.3 Renew

### 8.3.1 Request

IDCs maintain subscriptions by periodically sending *Renew* messages to the notification service prior to the expiration of a subscription. If a Renew message is not received prior to the expiration of a subscription then a new subscription must be created to continue exchanging *Notify* messages. The format of a request is provided below:

```
<xsd:element name="Renew">
  <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="TerminationTime"
      type="wsnt:AbsoluteOrRelativeTimeType "
      nillable="true" minOccurs="1" maxOccurs="1" />
     <xsd:element ref="wsnt:SubscriptionReference"
      minOccurs="1" maxOccurs="1" />
     <xsd:any namespace="##other" processContents="lax"
      minOccurs="0" maxOccurs="unbounded"/>
```

```
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:Renew

The container element for the renewal parameters\

/wsnt:Renew/wsnt:TerminationTime

A field suggesting a new TerminationTime for the subscription.

/wsnt:Renew/wsnt:SubscriptionReference

Indicates which subscription to renew. This field MUST match the field of the same name provided in the *SubscribeResponse* message.

### 8.3.2 Response

A successful response is provided below:

```
<xsd:element name="RenewResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsnt:TerminationTime"
       minOccurs="1" maxOccurs="1" />
      <xsd:element ref="wsnt:CurrentTime"
       minOccurs="0" maxOccurs="1" />
      <xsd:element ref="wsnt:SubscriptionReference"
       minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:RenewResponse

A container for response parameters.

/wsnt:RenewResponse/wsnt:TerminationTime

The new expiration of the subscription

/wsnt:RenewResponse/wsnt:CurrentTime

The time the response was sent.

/wsnt:RenewResponse/wsnt:SubscriptionReference

Indicates which subscription was renewed. This field MUST match the field of the same name provided in the *SubscribeResponse* message and the *Renew* message.

### 8.3.3 Example

An example of Domain 1 renewing its subscription to Domain 2 is shown below:

```
<s:Envelope ... >
  <s:Header>
    <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:Renew>
      <wsnt:TerminationTime xmlns:xsi="http://www.w3.org/
        2001/XMLSchema-instance" xsi:nil="1">
      <wsnt:SubscriptionReference>
        <wsa:Address>
          https://domain2.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
            <idc:subscriptionId>
              urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
            </idc:subscriptionId>
        </wsa:ReferenceParameters>
      </wsnt:SubscriptionReference>
    </wsnt:Renew>
  </s:Body>
</s:Envelope>
```

And the response is as follows:

```
<s:Envelope ... >
  <s:Header>
    <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:RenewResponse>
      <wsnt:TerminationTime>
        2009-10-01T16:02:30.000+00:00
      </wsnt:TerminationTime>
      <wsnt:CurrentTime>
        2009-10-01T15:02:30.000+00:00
```

```
      </wsnt:CurrentTime>
      <wsnt:SubscriptionReference>
        <wsa:Address>
          https://domain2.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
            <idc:subscriptionId>
              urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
            </idc:subscriptionId>
        </wsa:ReferenceParameters>
      </wsnt:SubscriptionReference>
    </wsnt:RenewResponse >
  </s:Body>
</s:Envelope>
```

# 9 Resource Scheduling

A dynamic circuit uses network resources (such as bandwidth) along a path. A *reservation* is created when a path with the desired resources for a circuit is found and reserved. *Resource scheduling* is the process by which reservations are created, modified, and cancelled. The IDC protocol defines operations for each of these functions.

## 9.1 Creating a Reservation

### 9.1.1 createReservation

The *createReservation* message is used to request the creation of a new reservation. It is worth noting that the base *resCreateContent* data type used contains only administrative information (see section **7.1**); all technology-specific information is contained in the pathInfo sub-object (see section **7.2**). The format of the request message is shown below:

```
<xsd:element name="createReservation"
  type="tns:resCreateContent" />
<xsd:complexType name="resCreateContent">
  <xsd:sequence>
    <xsd:element name="globalReservationId" type="xsd:string"
      maxOccurs="1" minOccurs="0"/>
    <xsd:element name="startTime" type="xsd:long" />
    <xsd:element name="endTime" type="xsd:long" />
    <xsd:element name="bandwidth" type="xsd:int" />
```

```
    <xsd:element name="description" type="xsd:string" />
    <xsd:element name="pathInfo" type="tns:pathInfo" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:createReservation

Container element included in the SOAP [SOAP] body of a message that contains the parameters for creating the reservation.

/idc:createReservation/idc:globalReservationId

MAY be included. It is used to uniquely identify the reservation across all IDCs. If omitted, the message recipient MUST generate an appropriately unique identifier and return it in the response. Typical use is that an end-user omits this field, and their home IDC instance creates a string with a format of *idc_id-seq_nr*

/idc:createReservation/idc:startTime and idc:createReservation/dc:endTime

MUST be included, and define the period for which the requested resources will be reserved. The field format is seconds-since-epoch.

/idc:createReservation/idc:bandwidth

MUST be included, and specifies the number of Mbps requested to be reserved.

/idc:createReservation/idc:description

MUST be included, and is a human-readable field meant to describe the purpose of the reservation.

/idc:createReservation/idc:pathInfo

MUST be included, and is extensively described in section **7.2**. The path information here represents what the reservation requester is asking from the IDC. The hops it contains may be domain, node, port, or link elements of id references.

### 9.1.2 createReservationResponse

The IDC returns a *createReservationResponse* after receiving a request and verifying its parameters are valid. The response indicates the reservation is in the ACCEPTED state. This means that the reservation has NOT been scheduled and no hold on resources yet exists. Instead it indicates that the reservation will be considered for further processing. The format of the message is below:

```
<xsd:element name="createReservationResponse"
  type="tns:createReply" />
<xsd:complexType name="createReply">
  <xsd:sequence>
```

```
    <xsd:element name="globalReservationId" type="xsd:string" />
    <xsd:element name="token" type="xsd:string" maxOccurs="1"
      minOccurs="0"/>
    <xsd:element name="status" type="xsd:string" />
    <xsd:element name="pathInfo" type="tns:pathInfo"
      maxOccurs="1" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:createReservationResponse

Container element included in the SOAP [SOAP] body of a message with the response of a *createReservation* operation.

/idc:createReservationResponse/idc:globalReservationId

MUST be included. Typically an IDC instance generates a string with a format of *idc_id-seq_nr* and returns it to the user through this field.

/idc:createReservationResponse/idc:token

MAY be included, and contains a token that is to be used during path signaling. The specific use cases for tokens are the subject of ongoing research.

/idc:createReservationResponse/idc:status

MUST contain the value ACCEPTED to indicate the request was received and the parameters were valid.

/idc:createReservationResponse/idc:pathInfo

MUST be included, and is extensively described in section **7.2**. This path information describes the path the IDC decided that the reservation will actually take.

### 9.1.3 RESERVATION_CREATE_CONFIRMED event

After the *createReservation* request has been forwarded to the last domain in the signaling path, the last domain can make final decisions about the path and resources that will be reserved and throw a RESERVATION_CREATE_CONFIRMED event. This takes the form of a *wsnt:Notify* message containing an *idc:event* element. Each domain received the event, makes final decision about local resources, and passes it back toward the first domain. See section 2.2.1 Resource Scheduling Chain for more information on this ordering. A RESERVATION_CREATE_CONFIRMED event MUST also meet the following requirements in addition to the general requirement of a Notify message:

- The */idc:event/idc:eventType* field MUST have a value of RESERVATION_CREATE_CONFIRMED
- The event MUST contain an */idc:event/idc:resDetails* object describing the reservation being created.
- The */idc:event/idc:resDetails/idc:pathInfo/idc:path* MUST contain full *nmwg-cp:link* elements for all hops local to the domain throwing the event and all domains toward the last domain (inclusive). This means the last domain must fill-in the hops for itself, the second-to-last domain must fill-in its hop plus pass the filled-in hops of the last domain, etc. By the time the first domain is reached all domains beyond the first domain should have full *nmwg-cp:link* elements.
- Each domain MUST select the resources that will be used in its domain and include those in the event. For example, in a reservation using VLANs a domain must select its VLANs and include that decision in the message.

### 9.1.4 RESERVATION_CREATE_COMPLETED event

After the first domain receives a RESERVATION_CREATE_CONFIRMED event it finalizes its resources and throws a RESERVATION_CREATE_COMPLETED event toward the end client and the next IDC in the signaling path. The next IDC is responsible for continuing this message along the chain and the message will eventually reach the last domain. When each domain receives the RESERVATION_CREATE_COMPLETED the MUST change the reservation state to PENDING. A domain MUST NOT make any changes to the resources selected when receiving a RESERVATION_CREATE_COMPLETED event. This event is for informational purposes only so that each domain knows the resource selected by other domains and thus has an equivalent view of the inter-domain reservation. Below are the message requirements specific to the RESERVATION_CREATE_COMPLETED event:

- The */idc:event/idc:eventType* field MUST have a value of RESERVATION_CREATE_COMPLETED
- The event MUST contain an */idc:event/idc:resDetails* object describing the reservation being created.
- The */idc:event/idc:resDetails/idc:pathInfo/idc:path* MUST contain full *nmwg-cp:link* elements for every hop in the inter-domain path.

## 9.2 Modifying a Reservation

### 9.2.1 modifyReservation

This message is used to initiate the modification of an existing PENDING or ACTIVE reservation. When the process completes either successfully or unsuccessfully the reservation MUST return to the state it was in prior to modification unless a network failure prevents this from happening. The modification message supports changes in bandwidth, start and end time, description, as well as path information. The user provides the Global Reservation Identifier of the reservation they wish to modify, as well as the desired new values of the parameters. The message recipient MAY accept all, some, or none of the new values depending on policy and user authorization.

The request message is described below:

```
<xsd:element name="modifyReservation"
  type="tns:modifyResContent" />
<xsd:complexType name="modifyResContent">
  <xsd:sequence>
    <xsd:element name="globalReservationId" type="xsd:string"
      maxOccurs="1" minOccurs="1"/>
    <xsd:element name="startTime" type="xsd:long" />
    <xsd:element name="endTime" type="xsd:long" />
    <xsd:element name="bandwidth" type="xsd:int" />
    <xsd:element name="description" type="xsd:string" />
    <xsd:element name="pathInfo" type="tns:pathInfo" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:modifyReservation

Container element included in the SOAP [SOAP] body of a message that contains the parameters for mofifying the reservation.

/idc:modifyReservation/idc:globalReservationId

MUST be included. It is used to specify the reservation to be modified.

/idc:modifyReservation/idc:startTime and /idc:modifyReservation/idc:endTime

MUST be included, and define the new period for which the requested resources will be reserved. The field format is seconds-since-epoch.

/idc:modifyReservation/idc:bandwidth

MUST be included, and specifies the new number of Mbps requested.

/idc:modifyReservation/idc:description

MUST be included, is the new human-readable field that describes the purpose of the reservation.

/idc:modifyReservation/idc:pathInfo

MUST be included, and is extensively described in section **7.2**. The path information here will replace the existing path information.

### 9.2.2 modifyReservationResponse

The *modifyReservationResponse* message gets returned after validating the parameters but prior to processing the request and setting the state to INMODIFY. It has the following format:

```
<xsd:element name="modifyReservationResponse"
  type="tns:modifyResReply" />
<xsd:complexType name="modifyResReply">
  <xsd:sequence>
    <xsd:element name="reservation" type="tns:resDetails" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:modifyReservationResponse

Container element included in the SOAP [SOAP] body of a message with the response of a *modifyReservation* operation.

idc:modifyReservationResponse/idc:resDetails

MUST be included, and is the full reservation description as it is after the changes the user requested. The data type is fully described in section **7.1**.

### 9.2.3 RESERVATION_MODIFY_CONFIRMED event

After the *modifyReservation* request has been forwarded to the last domain in the signaling path, the last domain can make final decisions about the modified path and resources that will be reserved and throw a RESERVATION_MODIFY_CONFIRMED event. This takes the form of a *wsnt:Notify* message containing an *idc:event* element. Each domain receives the event, makes final decision about local resources, and passes it back toward the first domain. See section 2.2.1 Resource Scheduling Chain for more information on the messahe ordering. A RESERVATION_MODIFY_CONFIRMED event meets the following requirements in addition to the general requirement of a Notify message:

- The */idc:event/idc:eventType* field MUST have a value of RESERVATION_MODIFY_CONFIRMED
- The event MUST contain an */idc:event/idc:resDetails* object describing the reservation being modified.
- The */idc:event/idc:resDetails/idc:pathInfo/idc:path* MUST contain full *nmwg-cp:link* elements for all hops local to the domain throwing the event and all domains toward the last domain (inclusive). This means the last domain must fill-in the hops for itself, the second-to-last domain must fill-in its hop plus pass the filled-in hops of the last domain, etc. By the time the first domain is reached all domains beyond the first domain should have full *nmwg-cp:link* elements of the reservation to be modified.
- An IDC MUST select the resources that will be used in the local domain after modification and include those in the event. For example, in a reservation using VLANs in a domain must select its VLANs and include that decision in the message.

### 9.2.4 RESERVATION_MODIFY_COMPLETED event

After the first domain receives a RESERVATION_MODIFY_CONFIRMED event it finalizes its resources and throws a RESERVATION_MODIFY_COMPLETED event toward the end client and the next IDC in the signaling path. The next IDC is responsible for continuing this message along the chain and the message will eventually reach the last domain. When each domain receives the RESERVATION_MODIFY_COMPLETED event it MUST change the reservation state back to PENDING or ACTIVE. A domain MUST NOT make any changes to the resources selected when receiving a RESERVATION_MODIFY_COMPLETED event. This event is for informational purposes only so that each domain knows the resource selected by other domains and thus has an equivalent view of the inter-domain reservation. Below are the message requirements specific to the RESERVATION_MODIFY_COMPLETED event:

- The */idc:event/idc:eventType* field MUST have a value of RESERVATION_ MODIFY _COMPLETED
- The event MUST contain an */idc:event/idc:resDetails* object describing the reservation that was modified.
- The */idc:event/idc:resDetails/idc:pathInfo/idc:path* MUST contain full *nmwg-cp:link* elements for every hop in the inter-domain path.

## 9.3 Cancelling a Reservation

### 9.3.1 cancelReservation

Cancellation of a reservation is used to release the resources held by a reservation including the removal of ACTIVE reservations before the originally defined end time. If a reservation is PENDING when a cancellation is received then only the resources need to be released and no circuit teardown is required. When the process completes the reservation enters the CANCELLED state. The request message to initate the cancellation process is described below:

```
<xsd:element name="cancelReservation"
type="tns:globalReservationId" />
<xsd:complexType name="globalReservationId">
  <xsd:sequence>
    <xsd:element name="gri" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:cancelReservation

Container element included in the SOAP [SOAP] body of a message with the cancellation parameters.

/idc:cancelReservation/idc:gri

MUST be included. The identifier for the reservation to be cancelled.

### 9.3.2 cancelReservationResponse

The response message of this operation is described below:

```
<xsd:element name="cancelReservationResponse" type="xsd:string"
/>
```

/idc:cancelReservationResponse

MUST be included; a human-readable string such as “Cancellation accepted”.

### 9.3.3 RESERVATION_CANCEL_CONFIRMED event

A RESERVATION_CANCEL_CONFIRMED event is first thrown when the last domain in the signaling path has released the necessary resources and removed the circuit from the network (if the reservation was ACTIVE). Each domain throws the RESERVATION_CANCEL_CONFIRMED method after receiving this event, removing the local circuit and releasing the resources until the first domain is reached. The only

additional requirements beyond those other events are that the event type is set to RESERVATION_CANCEL_CONFIRMED and the GRI is included.

### 9.3.4 RESERVATION_CANCEL_COMPLETED event

The RESRVATION_CANCEL_COMPLETED event is thrown after all domains have confirmed the reservation cancellation. The first domain throws it to the end-user and the next domain in the signaling path where its forwarded down the chain. The RESERVATION_CANCEL_COMPLETED message indicates that each domain can set the reservation status to CANCELLED. The only additional requirements beyond those for other events are that the event type is set to RESERVATION_CANCEL_COMPLETED and the GRI is included.

# 10 Signaling

Signaling is the process that triggers the creation of a reservation's circuit on the network. After a reservation is placed, a circuit with the reserved resources will not be created until signaling occurs. In addition to circuit creation, signaling also encompasses circuit refreshing and removal. Signaling may occur automatically or in response to messages received by the IDC. The type of signaling that occurs is indicated by the <idc:pathSetupMode> field specified in the *createReservation* message sent during resource scheduling (see section **5**).This section details the use of each signaling type.

## 10.1 Automatic vs Manual Signaling

A <idc:pathSetupMode> value of *timer-automatic* indicates that a circuit will be created at the reservation start-time and removed at the reservation end-time. Beyond resource scheduling, no message exchange is required between a requester of a *timer-automatic* reservation and the IDC that received the request. This type of signaling is useful in many cases but does have some implications. It requires that a requester either assume a circuit is created/removed at the specified time or continuously send *queryReservation* messages to get the circuit status (see section **11.2**). End-users or IDCs wishing to have more direct control over a circuit may want to consider using message signaling.

A reservation with <idc:pathSetupMode> set to *signal-xml* indicates that a circuit will only be created/removed upon receiving a signaling message. Signaling with messages is useful for those cases in which the requester wants more direct control over circuit instantiation beyond just creation at the start-time and removal at the end-time.

## 10.2 Creating a Circuit

### 10.2.1 Manually creating a circuit with *createPath*

If automatic signaling is specified then this message is NOT required. After a reservation has been placed requiring message signaling a circuit will not be created until the start time is reached AND a circuit creation message is received by the IDC. Circuit creation is signaled using the *createPath* operation. A *createPath* request is described in detail below:

```
<xsd:element name="createPath" type="tns:createPathContent" />
<xsd:complexType name="createPathContent">
  <xsd:sequence>
<xsd:element name="globalReservationId" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:createPath

A container element with parameters for creating the circuit. If the message is from the end-user then this element will be contained directly within the body of a SOAP message (see section **5**). If this element is passing between IDCs it will be encapsulated in an <idc:forward> element (see section **6**).

/idc:createPath/idc:globalReservationId

A required field indicating the global reservation identifier (GRI) of the reservation with the resources to instantiate.

The response of a *createPath* operation contains the following elements:

```
<xsd:element name="createPathResponse"
  type="tns:createPathResponseContent" />
<xsd:complexType name="createPathResponseContent">
  <xsd:sequence>
    <xsd:element name="globalReservationId" type="xsd:string" />
    <xsd:element name="status" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:createPathResponse

A container element returning the results of the *createPath* request. If the message is to the end-user then this element will be contained directly within the body of a SOAP message (see section **5**). If this element is passing between

IDCs it will be encapsulated in an <idc:forwardResponse> element (see section **6**).

/idc:createPathResponse /idc:status
: The status that resulted from the operation. It should have a value of INCREATE.

/idc:createPathResponse /idc:globalReservationId
: A required field indicating the global reservation identifier (GRI) of the reservation with the circuit that was created.

An IDC MUST complete the following order of tasks when processing the requests and responses of a *createPath* operation:

1. Upon receiving a *createPath* message the IDC should validate the parameters and immediate send a *createPathResponse* back to the requester
2. The IDC should next send an <idc:forward> message containing a <idc:createPath> element in the payload to the IDC of the next domain in the reservation's path. If there is no next domain in the path then the IDC should proceed to step 3.
3. Upon receiving a successful response from the IDC contacted in step 2, the IDC should contact the domain controller (DC) to create the local domain's portion of the circuit. Events should be triggered as described in the sections that follow.

### 10.2.2 UPSTREAM_PATH_SETUP_CONFIRMED event

Regardless of whether a circuit is signaled automatically or manually it must throw certain events. The UPSTREAM_PATH_SETUP_CONFIRMED event is thrown in a *wsnt:Notify* messaging containing an *idc:event* object. It is triggered in one of the following cases:

1. The local domain is the **first** domain in the path and it has completed circuit creation in the local domain.
2. The local domain has completed its local configuration AND it has received an UPSTREAM_PATH_SETUP_CONFIRMED event.

The *idc:event/idc:eventType* must be set to UPSTREAM_PATH_SETUP_CONFIRMED and it must contain the message must contain GRI of the reservation.

### 10.2.3 DOWNSTREAM_PATH_SETUP_CONFIRMED event

The DOWNSTREAM_PATH_SETUP_CONFIRMED event is thrown in a *wsnt:Notify* messaging containing an *idc:event* object. It is triggered in one of the following cases:

1. The local domain is the **last** domain in the path and it has completed circuit creation in the local domain.
2. The local domain has completed its local configuration AND it has received a DOWNSTREAM_PATH_SETUP_CONFIRMED event.

The *idc:event/idc:eventType* must be set to DOWNSTREAM_PATH_SETUP_CONFIRMED and it must contain the message must contain GRI of the reservation.

### 10.2.4 PATH_SETUP_COMPLETED event

The PATH_SETUP_COMPLETED event indicates that a reservation is created in every domain along the signaling path. The PATH_SETUP_COMPLETED event MUST be sent after a domain receives an UPSTREAM_PATH_SETUP_CONFIRMED event AND a DOWNSTREAM_PATH_SETUP_CONFIRMED event. The reservation is considered ACTIVE upon the triggering of this event. The *idc:event/idc:eventType* must be set to PATH_SETUP_COMPLETED and it must contain the message must contain GRI of the reservation.

## 10.3 Removing a circuit

### 10.3.1 Manually removing a circuit with *teardownPath*

This message is NOT required for manual signaling. When a circuit is no longer needed an end-user or IDC may send a *teardownPath* message to remove a circuit from the data plane. This message is different from *cancelReservation* (see section **9.3**) in that it does not remove a reservation's hold on network resources. This means that a circuit may be instantiated again after a *teardownPath* completes if another *createPath* message is sent before the reservation end time. A circuit MUST be removed from the data plane at reservation end time whether a *teardownPath* message is received or not. The *teardownPath* request is described below:

```
<xsd:element name="teardownPath"
  type="tns:teardownPathContent" />
<xsd:complexType name="teardownPathContent">
  <xsd:sequence>
    <xsd:element name="globalReservationId" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:teardownPath

A container element with parameters for tearing down the circuit. If the message is from the end-user then this element will be contained directly within the body of a SOAP message (see section **5**). If this element is passing between IDCs it will be encapsulated in an <idc:forward> element (see section **6**).

/idc:teardownPath/idc:globalReservationId
A required field indicating the global reservation identifier (GRI) of the reservation with the circuit to remove.

The response of a *teardownPath* operation is as follows:

```
<xsd:element name="teardownPathResponse"
  type="tns:teardownPathResponseContent" />
<xsd:complexType name="teardownPathResponseContent">
  <xsd:sequence>
    <xsd:element name="globalReservationId" type="xsd:string"/>
    <xsd:element name="status" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:teardownPathResponse
A container element returning the results of the *teardownPath* request. If the message is to the end-user then this element will be contained directly within the body of a SOAP message (see section **5**). If this element is passing between IDCs it will be encapsulated in an <idc:forwardResponse> element (see section **6**).

/idc:teardownPathResponse/idc:status
The status that resulted from the operation. It should have a value of INTEARDOWN.

/idc:teardownPathResponse/idc:globalReservationId
A required field indicating the global reservation identifier (GRI) of the circuit that was removed.

### 10.3.2 UPSTREAM_PATH_TEARDOWN_CONFIRMED event

The UPSTREAM_PATH_TEARDOWN_CONFIRMED event is thrown in a *wsnt:Notify* messaging containing an *idc:event* object. It is triggered in one of the following cases:

1. The local domain is the **first** domain in the path and it has completed circuit removal in the local domain.

2. The local domain has completed its local configuration AND it has received an UPSTREAM_PATH_TEARDOWN_CONFIRMED event.

The *idc:event/idc:eventType* must be set to UPSTREAM_PATH_TEARDOWN_CONFIRMED and it must contain the message must contain GRI of the reservation.

### 10.3.3 DOWNSTREAM_PATH_TEARDOWN_CONFIRMED event

The DOWNSTREAM_PATH_TEARDOWN_CONFIRMED event is thrown in a *wsnt:Notify* messaging containing an *idc:event* object. It is triggered in one of the following cases:

3. The local domain is the **last** domain in the path and it has completed circuit removal in the local domain.
4. The local domain has completed its local configuration AND it has received a DOWNSTREAM_PATH_TEARDOWN_CONFIRMED event.

The *idc:event/idc:eventType* must be set to DOWNSTREAM_PATH_TEARDOWN_CONFIRMED and it must contain the message must contain GRI of the reservation.

### 10.3.4 PATH_TEARDOWN_COMPLETED event

The PATH_TEARDOWN_COMPLETED event indicates that a reservation is removed in every domain along the signaling path. The PATH_TEARDOWN_COMPLETED event MUST be sent after a domain receives an UPSTREAM_PATH_TEARDOWN_CONFIRMED event AND a DOWNSTREAM_PATH_TEARDOWN_CONFIRMED event. The reservation is considered PENDING, CANCELLED, or FINISHED upon the triggering of this event and what caused the teardown. The *idc:event/idc:eventType* must be set to PATH_TEARDOWN_COMPLETED and it must contain the message must contain GRI of the reservation.

# 11 Polling Circuit Information

The IDC protocol currently provides two messages for finding information about reservations, one giving a summary list according to a number of search terms, and one providing reservation details given a global reservation identifier (GRI).

## 11.1 Listing Reservations

The *listReservations* operation returns a list of reservation that match a specified set of search parameters. The summary list as retrieved from a given IDC does not include intra-domain information that may be available from other IDCs along a reservation's path. All elements in a *listReservations* request MAY be included, and are used as either terms to limit the search, or to control the number of results returned. Search term elements can be combined to yield a subset of all stored reservations. The request for the *listReservations* operation is described below.

```
<xsd:element name="listReservations" type="tns:listRequest" />
<xsd:complexType name="listRequest">
  <xsd:sequence>
    <xsd:element name="resStatus" type="xsd:string"
      maxOccurs="5" minOccurs="0" />
    <xsd:sequence maxOccurs="1" minOccurs="0">
      <xsd:element name="startTime" type="xsd:long" />
      <xsd:element name="endTime" type="xsd:long" />
    </xsd:sequence>
    <xsd:element name="description" type="xsd:string"
       maxOccurs="1" minOccurs="0" />
    <xsd:element name="linkId" type="xsd:string"
      maxOccurs="unbounded" minOccurs="0" />
    <xsd:element name="vlanTag" type="tns:vlanTag"
       minOccurs="0" maxOccurs="unbounded" />
    <xsd:element name="resRequested" type="xsd:int"
       minOccurs="0"/>
    <xsd:element name="resOffset" type="xsd:int"
       minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

/idc:listReservations

Element included in the body of a SOAP [SOAP] message of type idc:listRequest that contains search constraints for a desired list of reservations.

/idc:listReservations/idc:resStatus

Contains a list of statuses to constrain the search. It may include 0 or all of the following: ACTIVE, PENDING, FINISHED, CANCELLED, and FAILED. If it is not given, reservations with all statuses are returned, depending on the other search

parameters. If one or more are given, only reservations with those statuses are returned.

/idc:listReservations/idc:startTime
: Constrains the search such that only reservations ending after the start time are returned.

/idc:listReservations/idc:endTime
: Constrains the search such that only reservations starting before the end time are returned.

/idc:listReservations/idc:description
: Constrains the search such that only those reservations with that string in their descriptions are returned.

/idc:listReservations/idc:linkId
: Contains a list of zero or more link ids. Constrains the search such that only reservations with those link ids in their intradomain paths are returned.

/idc:listReservations/idc:vlanTag
: Contains a list of zero or more VLAN tags. Constrains the search such that only reservations with those VLAN tags are returned.

/idc:listReservations/idc:resRequested
: Contains an integer indicating how many results are returned in one request.

/idc:listReservations/idc:resOffset
: Contains an integer offset into the total set of reservations. Taken together with resRequested, it can be used to page through the results.

The response to a *listReservations* operation contains the following elements:

```
<xsd:element name="listReservationsResponse"
    type="tns:listReply" />
<xsd:complexType name="listReply">
  <xsd:sequence>
    <xsd:element name="resDetails" type="tns:resDetails"
      maxOccurs="unbounded" minOccurs="0" />
    <xsd:element name="totalResults" type="xsd:int"
      minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:listReservationsResponse

Element included in the body of a SOAP [SOAP] message of type idc:listReply that contains zero or more objects with a summary of each reservation that matched the search constraints in the *listReservations* request.

/idc:listReservationsResponse/idc:resDetails
: Zero or more idc:resDetails instances (see section **4.1**) containing information about the reservations satisfying the search criteria, and may be empty.

/idc:listReservationsResponse/idc:totalResults
: An optional element containing the number of instances returned.

### 11.1.1 Example

The following are examples of a *listReservations* request and response.

In the following request, reservations are requested that have finished successfully and have a VLAN tag of 3000.

```
<soap:Envelope ...>
<soap:Body>
  <idc:listReservations>
    <idc:resStatus>FINISHED</idc:resStatus>
    <idc:vlanTag tagged="true">3000</idc:vlanTag>
    <idc:resRequested>10</idc:resRequested>
    <idc:resOffset>0</idc:resOffset>
  </idc:listReservations>
</soap:Body>
</soap:Envelope>
```

An abstracted view of the response is below:

```
<soap:Envelope ...>
<soap:Body>
  <idc:listReservationsResponse>
    <idc:resDetails>
      <idc:globalReservationId>domain1.net-1
      </idc:globalReservationId>
      <idc:login>user@domain.net</idc:login>
      <idc:status>FINISHED</idc:status>
      <idc:startTime>1206486746</idc:startTime>
      <idc:endTime>1206486962</idc:endTime>
      <idc:createTime>1206486752</idc:createTime>
      <idc:bandwidth>25</idc:bandwidth>
      <idc:description>default layer 2 test
       reservation</idc:description>
      <idc:pathInfo>
```

```
        <idc:pathSetupMode>timer-automatic</idc:pathSetupMode>
        <idc:path id="unimplemented">
          <ctrlp:hop id="hop1">
            <linkIdRef>linkId1</linkIdRef>
          </hop>
          ...
          <ctrlp:hop id="hopN">
            <linkIdRef>linkIdN</linkIdRef>
          </hop>
        </idc:path>
      </idc:pathInfo>
      <idc:layer2Info>
        <idc:srcVtag tagged="true">3000</idc:srcVtag>
        <idc:destVtag tagged="true">3000</idc:destVtag>
        <idc:srcEndpoint>srcLinkId</idc:srcEndpoint>
        <idc:destEndpoint>destLinkId</idc:destEndpoint>
      </idc:layer2Info>
    </idc:resDetails>
....
    <idc:totalResults><12></idc:totalResults>
  </idc:listReservationsResponse>
</soap:Body>
</soap:Envelope>
```

## 11.2 Querying Reservations

The *queryReservation* operation returns details about a specified reservation. The *queryReservation* operation MAY be forwarded to other domains to obtain additional information about the requested reservation. The request for the *queryReservation* operation is described below.

```
<xsd:element name="queryReservation"
type="tns:globalReservationId" />
<xsd:complexType name="globalReservationId">
        <xsd:sequence>
            <xsd:element name="gri" type="xsd:string" />
        </xsd:sequence>
 </xsd:complexType>
```

/idc:queryReservation

Element included in the body of a SOAP [SOAP] message that contains information to identify the reservation to query.

/idc:queryReservation/idc:globalReservationId

The unique global reservation id (GRI) of the reservation to query.

The following is the response to the queryReservation request:

```
<xsd:element name="queryReservationResponse"
  type="tns:resDetails" />
```

/idc:queryReservationResponse

Element included in the body of a SOAP [SOAP] message that contains the details of a queried reservation.

/idc:queryReservationResponse/idc:resDetails

An instance (see section **4.1**), if any, with the given global reservation id, containing path information from all IDC's participating in the circuit.

### 11.2.1 Example

An example of a *queryReservation* operation is shown below:

```
<soap:Envelope ...>
<soap:Body>
  <idc:queryReservation>
    <idc:gri>domain1.net-1</idc:gri>
  </idc:queryReservation>
</soap:Body>
</soap:Envelope>
```

See the preceding section for an example of a resDetails instance that would be returned as part of a queryReservationResponse.

# 12 Topology Exchange

The inter-domain controller (IDC) currently offers limited support for exchanging topology information between domains. It defines one operation named *getNetworkTopology* that returns a view of the inter-domain topology. All topology elements are described using the NMWG Control Plane [NMWG-CP] topology schema. Topology exchange is still an area of active development and more sophisticated services will provide this function in the future. As a result this call may be deprecated in the future. The *getNetworkTopology* request is described below:

```
<xsd:element name="getNetworkTopology"
  type="tns:getTopologyContent" />
```

```
<xsd:complexType name="getTopologyContent">
  <xsd:sequence>
    <xsd:element name="topologyType" type="xsd:string"
minOccurs="1" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:getNetworkTopology

Container element for request parameters that is included directly in the body of a SOAP message.

/idc:getNetworkTopology/idc:topologyType

Required parameter indicating the topology view to return. Currently only the value *all* is supported which indicates that an IDC should return its own topology in its entirety.

The response to a *getNetworkTopology* request looks like the following:

```
<xsd:element name="getNetworkTopologyResponse"
  type="tns:getTopologyResponseContent" />
<xsd:complexType name="getTopologyResponseContent">
  <xsd:sequence>
   <xsd:element ref="nmwg-cp:topology" minOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

/idc:getNetworkTopologyResponse

Container element for the topology returned from an IDC.

/idc:getNetworkTopologyResponse/nmwg-cp:topology

The topology element as defined by the NMWG Control Plane [NMWG-CP] schema is the root element for a description of a network.

# 13 Brokered Notification

Implementations may optionally choose to have notification message distributed by a NotificationBroker. This section outlines the user of the messages defined by the WS-BrokeredNotification [WSNB] specification in the context of the IDC protocol.

## 13.1 RegisterPublisher

### 13.1.1 Request

The *RegisterPublisher* message is only required for cases where a NotificationBroker is used as specified by WS-BrokeredNotification [WSBN]. *RegisterPublisher* is sent from the IDC to the NotificationBroker to indicate that the IDC would like to distribute notifications through the broker. The NotificationBroker the registers the IDC by assign it an IDC to be used in all subsequent *Notify* message that it will distribute. The format of the message request sent by the IDC is shown below:

```
<xsd:element name="RegisterPublisher">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="PublisherReference"
        type="wsa:EndpointReferenceType"
        minOccurs="0" maxOccurs="1" />
      <xsd:element name="Topic"
        type="wsn-b:TopicExpressionType"
        minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="Demand" type="xsd:boolean"
        default="false" minOccurs="0" maxOccurs="1" />
      <xsd:element name="InitialTerminationTime"
        type="xsd:dateTime" minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt-br:RegisterPublisher

A container for the parameters used to register a publisher with the NotificationBroker

/wsnt-br:RegisterPublisher/wsnt:PublisherReference

A field containing a URL that identifies the publisher (IDC)

/wsnt-br:RegisterPublisher/wsnt:Topic

An optional list of topics that the IDC will publish. If not specified any topic will be allowed.

/wsnt-br:RegisterPublisher/wsnt:InitialTerminationTime

An optional suggestion of when the registration should expire.

### 13.1.2 Response

The format of the response sent from the NotificationBroker to the IDC is shown below:

```
<xsd:element name="RegisterPublisherResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="PublisherRegistrationReference"
        type="wsa:EndpointReferenceType"
        minOccurs="1" maxOccurs="1" />
      <xsd:element name="ConsumerReference"
        type="wsa:EndpointReferenceType"
        minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt-br:RegisterPublisherResponse

A container for the response

/wsnt-br:RegisterPublisherResponse/wsnt:PublisherRegistrationReference

An identifier of the registration created by the request. The identifier consists of the URL of the NotificationBroker where the IDC registered and a locally unique ID. See section **13.1.3** for more information.

/wsnt-br:RegisterPublisherResponse/wsnt:ConsumerReference

The URL of the NotificationBroker where the IDC must send *Notify* messages it wants distributed.

### 13.1.3 Identifying Publisher Registrations

A publisher registers with the NotificationBroker and creates a registration resource. This registration resource can be managed by using an identifier in a subsequent call. The identifier takes the format of a WS-Addressing [WSA] EndpointReferenceType where the URL in the *wsa:Address* field is that of the NotificationBroker. The *wsa:ReferenceParameters* must also contain an *idc:publisherRegistration* element that can take any form, such as a UUID, as long as its unique to the local domain.

```
<wsnt-br:PublisherRegistrationReference>
    <wsa:Address>
       https://mydomain.net/NotificationBroker
    </wsa:Address>
    <wsa:ReferenceParameters>
        <idc:publisherRegistrationId>
            urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
```

```
        </idc:publisherRegistrationId>
    </wsa:ReferenceParameters>
</wsnt-br:PublisherRegistrationReference>
```

### 13.1.4 Examples

In this example the Domain 1 IDC registers with a NotificationBroker also in Domain 1:

```
<s:Envelope ... >
  <s:Header>
   <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt-br:RegisterPublisher>
      <wsnt-br:PublisherReference>
        <wsa:Address>https://domain1.net/IDC</wsa:Address>
      </wsnt-br:PublisherReference>
    </wsnt-br:RegisterPublisher>
  </s:Body>
</s:Envelope>
```

The Domain 1 NotificationBroker then responds with the following:

```
<s:Envelope ... >
  <s:Header>
   <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt-br:RegisterPublisherResponse>
      <wsnt-br:PublisherRegistrationReference>
        <wsa:Address>
          https://domain1.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
          <idc:publisherRegistrationId>
            urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
          </idc:publisherRegistrationId>
        </wsa:ReferenceParameters>
      </wsnt-br:PublisherRegistrationReference>
      <wsnt-br:ConsumerReference>
        <wsa:Address>
          https://domain1.net/NotificationBroker
        </wsa:Address>
```

```
      </wsnt-br:ConsumerReference>
    </wsnt-br:RegisterPublisherResponse>
  </s:Body>
</s:Envelope>
```

## 13.2 DestroyRegistration

### 13.2.1 Request

If an IDC no longer wants to publish notifications to a NotificationBroker it can call the *DestroyRegistration* operation. The format of this call's request in the IDC context is as follows:

```
<xsd:element name="DestroyRegistration">
  <xsd:complexType>
    <xsd:sequence>
       <xsd:element name="PublisherRegistrationReference"
         type="wsa:EndpointReferenceType"
         minOccurs="1" maxOccurs="1" />
       <xsd:any namespace="##other" processContents="lax"
         minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute/>
  </xsd:complexType>
</xsd:element>
```

/wsnt-br:DestroyRegistration

Container for request

/wsnt-br:DestroyRegistration/wsnt-br:PublisherRegistrationReference

The identifier of the publisher registration resource to detroy. It MUST match the corresponding element included in the *RegisterPublisherResponse* message.

### 13.2.2 Response

```
<xsd:element name="DestroyRegistrationResponse">
  <xsd:complexType>
    <xsd:sequence>
       <xsd:element name="PublisherRegistrationReference"
         type="wsa:EndpointReferenceType"
         minOccurs="1" maxOccurs="1" />
       <xsd:any namespace="##other" processContents="lax"
         minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute/>
```

```
  </xsd:complexType>
</xsd:element>
```

/wsnt-br:DestroyRegistrationResponse
: Container for request

/wsnt-br:DestroyRegistration/wsnt-br:PublisherRegistrationReference
: The identifier of the publisher registration resource that was destroyed. It MUST match the corresponding element included in the *RegisterPublisherResponse* message.

### 13.2.3 Examples

The following example demonstrates the Domain 1 IDC destroying its registration with the Domain 1 NotificationBroker:

```
<s:Envelope ... >
  <s:Header>
   <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt-br:DestroyRegistration>
      <wsnt-br:PublisherRegistrationReference>
        <wsa:Address>
          https://domain1.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
          <idc:publisherRegistrationId>
            urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
          </idc:publisherRegistrationId>
        </wsa:ReferenceParameters>
      </wsnt-br:PublisherRegistrationReference>
    </wsnt-br:DestroyRegistration>
  </s:Body>
</s:Envelope>
```

The response returned by the NotificationBroker looks like the following:

```
<s:Envelope ... >
  <s:Header>
   <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt-br:DestroyRegistration>
      <wsnt-br:PublisherRegistrationReference>
```

```
        <wsa:Address>
          https://domain1.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
          <idc:publisherRegistrationId>
            urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
          </idc:publisherRegistrationId>
        </wsa:ReferenceParameters>
      </wsnt-br:PublisherRegistrationReference>
    </wsnt-br:DestroyRegistrationResponse>
  </s:Body>
</s:Envelope>
```

# 14 Advanced Subscription Management

In addition to the operations outlined in section **8** Notification Interface, WS-Notification also specifies calls for managing subscriptions. These calls are optional for an IDC but MAY be implemented as a convenience to non-IDC subscribers.

## 14.1 Unsubscribe

### 14.1.1 Request

The *Unsubscribe* operation expires a reservation prior to its termination time. It is not required between IDCs but can be useful for managing subscriptions. The format of the request is detailed below:

```
<xsd:element name="Unsubscribe">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsnt:SubscriptionReference"
       minOccurs="1" maxOccurs="1" />
     <xsd:any namespace="##other" processContents="lax"
      minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:Unsubscribe

The container element for the unsubscribe parameters

/wsnt:Unsubscribe /wsnt:SubscriptionReference

Indicates which subscription to cancel. This field MUST match the field of the same name provided in the *SubscribeResponse* message.

### 14.1.2 Response

A successful response is provided below:

```
<xsd:element name=" UnsubscribeResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsnt:SubscriptionReference"
       minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:UnsubscribeResponse

A container for the response parameters.

/wsnt:UnsubscribeResponse /wsnt:SubscriptionReference

Indicates which subscription was cancelled. This field MUST match the field of the same name provided in the *SubscribeResponse* message and the *Unsubscribe* message.

### 14.1.3 Example

An example of Domain 1 unsubscribing from Domain 2 notifications is shown below:

```
<s:Envelope ... >
  <s:Header>
    <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:Unsubscribe>
      <wsnt:SubscriptionReference>
        <wsa:Address>
          https://domain2.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
            <idc:subscriptionId>
              urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
            </idc:subscriptionId>
        </wsa:ReferenceParameters>
      </wsnt:SubscriptionReference>
    </wsnt:Unsubscribe>
  </s:Body>
</s:Envelope>
```

And the response is as follows:

```
<s:Envelope ... >
  <s:Header>
    <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:UnsubscribeResponse>
      <wsnt:SubscriptionReference>
        <wsa:Address>
          https://domain2.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
          <idc:subscriptionId>
            urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
          </idc:subscriptionId>
        </wsa:ReferenceParameters>
      </wsnt:SubscriptionReference>
    </wsnt:UnsubscribeResponse>
  </s:Body>
</s:Envelope>
```

## 14.2 PauseSubscription

### 14.2.1 Request

The *PauseSubscription* operation temporarily suspends the sending of notifications for a period of time. Notifications can later be resumed by sending a *ResumeSubscription* message prior to expiration. It is not required between IDCs but can be useful for managing subscriptions. The format of the request is detailed below:

```
<xsd:element name="PauseSubscription">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsnt:SubscriptionReference"
       minOccurs="1" maxOccurs="1" />
     <xsd:any namespace="##other" processContents="lax"
      minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:PauseSubscription

The container element for the pause parameters

/wsnt:PauseSubscription /wsnt:SubscriptionReference

Indicates which subscription to pause. This field MUST match the field of the same name provided in the *SubscribeResponse* message.

### 14.2.2 Response

A successful response is provided below:

```
<xsd:element name="PauseSubscriptionResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsnt:SubscriptionReference"
       minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:PauseSubscriptionResponse

A container for the response parameters.

/wsnt:PauseSubscriptionResponse /wsnt:SubscriptionReference

Indicates which subscription was paused. This field MUST match the field of the same name provided in the *SubscribeResponse* message and the *PauseSubscription* message.

### 14.2.3 Example

An example of Domain 1 pausing notifications from Domain 2 is shown below:

```
<s:Envelope ... >
  <s:Header>
    <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:PauseSubscription>
      <wsnt:SubscriptionReference>
        <wsa:Address>
          https://domain2.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
            <idc:subscriptionId>
              urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
```

```
            </idc:subscriptionId>
          </wsa:ReferenceParameters>
        </wsnt:SubscriptionReference>
      </wsnt:PauseSubscription >
    </s:Body>
</s:Envelope>
```

And the response is as follows:

```
<s:Envelope ... >
  <s:Header>
    <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:PauseSubscriptionResponse>
      <wsnt:SubscriptionReference>
        <wsa:Address>
          https://domain2.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
          <idc:subscriptionId>
            urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
          </idc:subscriptionId>
        </wsa:ReferenceParameters>
      </wsnt:SubscriptionReference>
    </wsnt:PauseSubscriptionResponse >
  </s:Body>
</s:Envelope>
```

## 14.3 ResumeSubscription

### 14.3.1 Request

The *ResumeSubscription* operation resumes the sending of notifications for a subscription that was previously paused. It is not required between IDCs but can be useful for managing subscriptions. The format of the request is detailed below:

```
<xsd:element name="ResumeSubscription">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsnt:SubscriptionReference"
       minOccurs="1" maxOccurs="1" />
     <xsd:any namespace="##other" processContents="lax"
      minOccurs="0" maxOccurs="unbounded"/>
```

```
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:ResumeSubscription

The container element for the resume parameters

/wsnt:ResumeSubscription /wsnt:SubscriptionReference

Indicates which subscription to resume. This field MUST match the field of the same name provided in the *SubscribeResponse* message.

### 14.3.2 Response

The format of a successful response is provided below:

```
<xsd:element name="ResumeSubscriptionResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsnt:SubscriptionReference"
       minOccurs="0" maxOccurs="1" />
      <xsd:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

/wsnt:ResumeSubscriptionResponse

A container for the response parameters.

/wsnt:ResumeSubscriptionResponse /wsnt:SubscriptionReference

Indicates which subscription was resumed. This field MUST match the field of the same name provided in the *SubscribeResponse* message and the *ResumeSubscription* message.

### 14.3.3 Example

An example of Domain 1 resume notifications from Domain 2 is shown below:

```
<s:Envelope ... >
  <s:Header>
    <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:ResumeSubscription>
      <wsnt:SubscriptionReference>
        <wsa:Address>
```

```
          https://domain2.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
            <idc:subscriptionId>
              urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
            </idc:subscriptionId>
        </wsa:ReferenceParameters>
      </wsnt:SubscriptionReference>
    </wsnt:ResumeSubscription >
  </s:Body>
</s:Envelope>
```

And the response is as follows:

```
<s:Envelope ... >
  <s:Header>
    <!-- WS-Security headers go here -->
  </s:Header>
  <s:Body>
    <wsnt:ResumeSubscriptionResponse>
      <wsnt:SubscriptionReference>
        <wsa:Address>
          https://domain2.net/NotificationBroker
        </wsa:Address>
        <wsa:ReferenceParameters>
          <idc:subscriptionId>
            urn:uuid:bca0c6f2-c408-482a-b7f5-bb936ae7ff92
          </idc:subscriptionId>
        </wsa:ReferenceParameters>
      </wsnt:SubscriptionReference>
    </wsnt:ResumeSubscriptionResponse >
  </s:Body>
</s:Envelope>
```

# 15 Appendix A: IDC Topics and Events

| Topic | Events |
|---|---|
| idc:IDC | RESERVATION_CREATE_CONFIRMED<br>RESERVATION_MODIFY_CONFIRMED<br>RESERVATION_CANCEL_CONFIRMED<br>DOWNSTREAM_PATH_SETUP_CONFIRMED<br>UPSTREAM_PATH_SETUP_CONFIRMED<br>DOWNSTREAM_PATH_TEARDOWN_CONFIRMED |

| | |
|---|---|
| | UPSTREAM_PATH_TEARDOWN_CONFIRMED<br>RESERVATION_CREATE_COMPLETED<br>RESERVATION_MODIFY_COMPLETED<br>RESERVATION_CANCEL_COMPLETED<br>PATH_SETUP_COMPLETED<br>PATH_TEARDOWN_COMPLETED<br>RESERVATION_CREATE_FAILED<br>RESERVATION_MODIFY_FAILED<br>RESERVATION_CANCEL_FAILED<br>PATH_SETUP_FAILED<br>PATH_REFRESH_FAILED<br>PATH_TEARDOWN_FAILED |
| idc:INFO | RESERVATION_CREATE_COMPLETED<br>RESERVATION_MODIFY_COMPLETED<br>RESERVATION_CANCEL_COMPLETED<br>PATH_SETUP_COMPLETED<br>PATH_REFRESH_COMPLETED<br>PATH_TEARDOWN_COMPLETED<br>RESERVATION_CREATE_FAILED<br>RESERVATION_MODIFY_FAILED<br>RESERVATION_CANCEL_FAILED<br>PATH_SETUP_FAILED<br>PATH_REFRESH_FAILED<br>PATH_TEARDOWN_FAILED |
| idc:DEBUG | RESERVATION_CREATE_RECEIVED<br>RESERVATION_CREATE_ACCEPTED<br>RESERVATION_CREATE_STARTED<br>RESERVATION_MODIFY_RECEIVED<br>RESERVATION_MODIFY_ACCEPTED<br>RESERVATION_MODIFY_STARTED<br>RESERVATION_CANCEL_RECEIVED<br>RESERVATION_CANCEL_ACCEPTED<br>RESERVATION_CANCEL_STARTED<br>PATH_SETUP_RECEIVED<br>PATH_SETUP_ACCEPTED<br>PATH_SETUP_STARTED<br>PATH_REFRESH_RECEIVED<br>PATH_REFRESH_ACCEPTED<br>PATH_REFRESH_STARTED<br>PATH_TEARDOWN_RECEIVED<br>PATH_TEARDOWN_ACCEPTED<br>PATH_TEARDOWN_STARTED<br>RESERVATION_LIST_RECEIVED<br>RESERVATION_LIST_STARTED<br>RESERVATION_LIST_COMPLETED |

| | |
|---|---|
| | RESERVATION_LIST_RETURNED<br>RESERVATION_QUERY_RECEIVED<br>RESERVATION_QUERY_STARTED<br>RESERVATION_QUERY_COMPLETED<br>RESERVATION_QUERY_RETURNED<br>RESERVATION_CREATE_FORWARD_STARTED<br>RESERVATION_CREATE_FORWARD_ACCEPTED<br>RESERVATION_MODIFY_FORWARD_STARTED<br>RESERVATION_MODIFY_FORWARD_ACCEPTED<br>RESERVATION_CANCEL_FORWARD_STARTED<br>RESERVATION_CANCEL_FORWARD_ACCEPTED<br>PATH_SETUP_FORWARD_STARTED<br>PATH_SETUP_FORWARD_ACCEPTED<br>PATH_REFRESH_FORWARD_STARTED<br>PATH_REFRESH_FORWARD_ACCEPTED<br>PATH_TEARDOWN_FORWARD_STARTED<br>PATH_TEARDOWN_FORWARD_ACCEPTED<br>RESERVATION_LIST_FORWARD_STARTED<br>RESERVATION_LIST_FORWARD_COMPLETED<br>RESERVATION_QUERY_FORWARD_STARTED<br>RESERVATION_QUERY_FORWARD_COMPLETED<br>RESERVATION_EXPIRES_IN_1DAY<br>RESERVATION_EXPIRES_IN_7DAYS<br>RESERVATION_EXPIRES_IN_30DAYS<br>RESERVATION_PERIOD_STARTED<br>RESERVATION_PERIOD_FINISHED<br>IDC_STARTED |
| idc:ERROR | IDC_FAILED<br>RESERVATION_CREATE_FAILED<br>RESERVATION_MODIFY_FAILED<br>RESERVATION_CANCEL_FAILED<br>PATH_SETUP_FAILED<br>PATH_REFRESH_FAILED<br>PATH_TEARDOWN_FAILED<br>LIST_RESERVATION_FAILED<br>QUERY_RESERVATION_FAILED |

# 16 Appendix B: The Meta-scheduler Model

The IDC protocol supports a meta-scheduler model of messaging. In the meta-scheduler model a centralized service, called a meta-scheduler, accepts an end-user's request then individually contacts each relevant domain's IDC. *Figure 16.1* shows a diagram describing the meta-scheduler model:

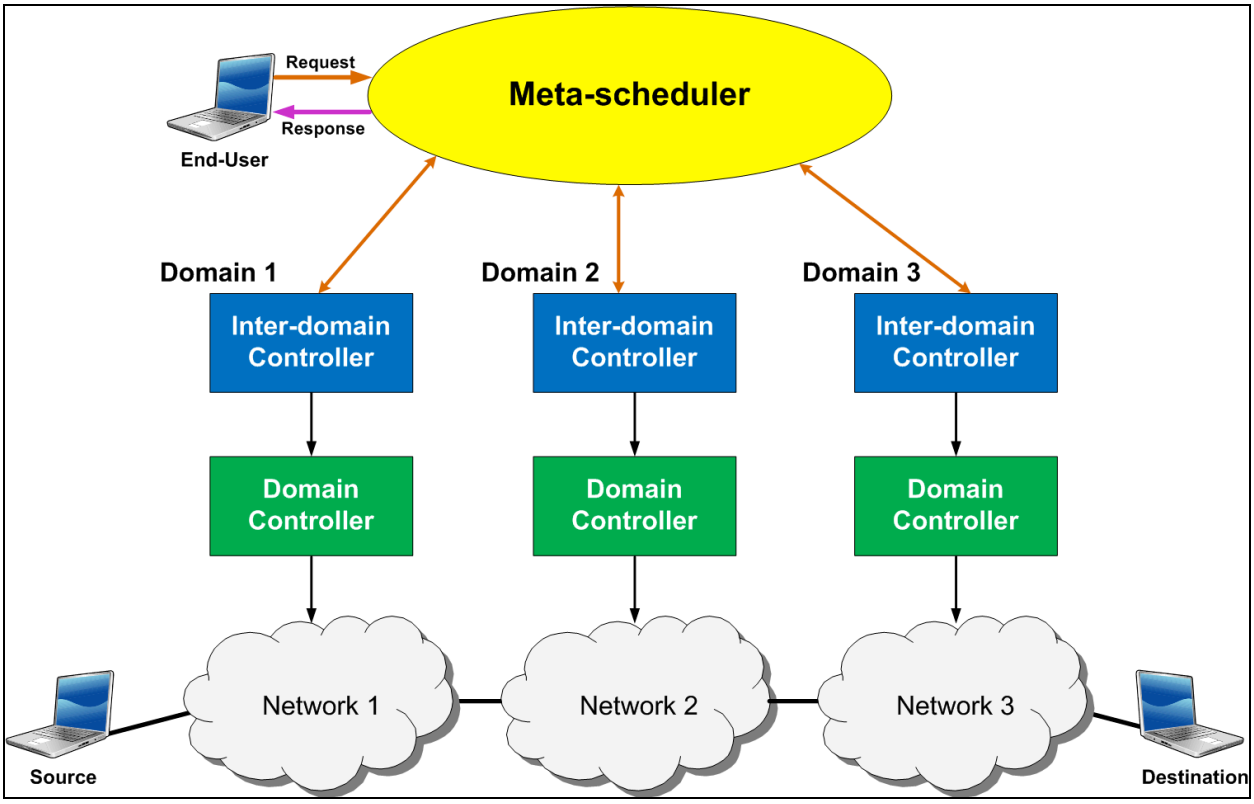


***Figure 16.1*** *An example of the meta-scheduler model with 3 domains*

In the figure an end-user sends a request to a meta-scheduler that involves resources on three domains. The meta-scheduler individually contacts the IDCs of domains 1,2, and 3 and no interaction occurs between IDCs. Each IDC returns the result of the request and the meta-scheduler aggregates their information and returns it to the end-user. The IDC protocol specifies some, but certainly not all, mechanisms to support this type of messaging model

The IDC protocol DOES NOT define an interface between the end-user and the meta-scheduler; however, the current version of this specification DOES provide limited support for interaction between meta-schedulers and IDCs. This can be achieved by sending the requests specified in the optional end-user SOAP interface as defined in section **5** of this document. Each of these requests sent by the meta-scheduler MUST currently only reference resources in the domain on which an IDC resides. The meta-scheduler is still an area of extensive research in this protocol, and support may be extended in future versions of the IDC protocol.

# 17 Appendix C: Create Reservation Example

This section contains an example of the messages sent and received by an end-user creating a reservation. The example demonstrates a request for a reservation between the source and destination displayed in *Figure 2.2*. The request message is shown below:

```
<soap:Envelope ...>
  <soap:Header>
    [end-user security credentials]
  </soap:Header>
  <soap:Body…>
    <idc:createReservation>
      <idc:startTime>1210847896</idc:startTime>
      <idc:endTime>1213847896</idc:endTime>
      <idc:bandwidth>1000</idc:bandwidth>
      <idc:description>1 Gbps example</idc:description>
      <idc:pathInfo>
         <idc:pathSetupMode>timer-automatic<idc:pathSetupMode>
         <idc:layer2Info>
         <idc:srcEndpoint>hostname.domain1.net<idc:srcEndpoint>
         <idc:destEndpoint>hostname.domain3.net<idc:destEndpoint>
        </idc:layer2Info>
      </idc:pathInfo>
    </idc:createReservation>
  </soap:Body>
</soap:Envelope>
```

After the Domain 1 IDC receives the above message and validates its parameters, converts the provided names to URNs using a lookup mechanism, and immediately returns the following response to the user:

```
<soap:Envelope ...>
  <soap:Body …>
    <idc:createReservationResponse>
      <idc:globalReservationId>domain1.net-
1</idc:globalReservationId>
      <idc:status>ACCEPTED</idc:status>
      <idc:pathInfo>
        <idc:pathSetupMode>timer-automatic<idc:pathSetupMode>
         <idc:layer2Info>
          <idc:srcEndpoint>
          urn:ogf:network:domain=domain1.net:node=1:port=1:link=1
          </idc:srcEndpoint>
```

```
          <idc:destEndpoint>
          urn:ogf:network:domain=domain3.net:node=2:port=1:link=1
          </idc:destEndpoint>
        </idc:layer2Info>
      </idc:pathInfo>
    </idc:createReservationResponse>
  </soap:Body>
</soap:Envelope>
```

Domain 1 then begins processing the reservation. It calculates a path that's not oversubscribed and determines which VLANs are available on the local links since the technology type of the requested link indicates VLANS should be used. It determines Domain 2 is the next domain in the path and sends the following *idc:forward* message to Domain 2's IDC:

```
<soap:Envelope ...>
  <soap:Body …>
   <idc:forward>
    <idc:payloadSender>user1</idc:payloadSender>
    <idc:payload>
     <idc:createReservation>
      <idc:globalReservationId>domain1.net-
1</idc:globalReservationId>
      <idc:startTime>1210847896</idc:startTime>
      <idc:endTime>1213847896</idc:endTime>
      <idc:bandwidth>1000</idc:bandwidth>
      <idc:description>1 Gbps example</idc:description>
      <idc:pathInfo>
        <idc:pathSetupMode>timer-automatic<idc:pathSetupMode>
        <idc:path>
         <nmwg-cp:hop id="1">
          <nmwg-cp:link id="
        urn:ogf:network:domain=domain1.net:node=1:port=1:link=1">
           <trafficEngineeringMetric>
             10
           </trafficEngineeringMetric>
           <SwitchingCapabilityDescriptors>
             <switchingcapType>l2sc</switchingcapType>
             <encodingType>ethernet</encodingType>
             <switchingCapabilitySpecificInfo>
               <interfaceMTU>9000</interfaceMTU>
               <vlanRangeAvailability>
                3220-3224
               </vlanRangeAvailability>
               <suggestedVLANRange>3221</suggestedVLANRange>
```

```
            </switchingCapabilitySpecificInfo>
          </SwitchingCapabilityDescriptors>
        </nmwg-cp:link>
      </nmwg-cp:hop>
      <nmwg-cp:hop id="2">
        <nmwg-cp:link id="
    urn:ogf:network:domain=domain1.net:node=2:port=1:link=1">
          <trafficEngineeringMetric>
            10
          </trafficEngineeringMetric>
          <SwitchingCapabilityDescriptors>
            <switchingcapType>l2sc</switchingcapType>
            <encodingType>ethernet</encodingType>
            <switchingCapabilitySpecificInfo>
              <interfaceMTU>9000</interfaceMTU>
              <vlanRangeAvailability>
               3220-3224
              </vlanRangeAvailability>
              <suggestedVLANRange>3221</suggestedVLANRange>
            </switchingCapabilitySpecificInfo>
          </SwitchingCapabilityDescriptors>
        </nmwg-cp:link>
      </nmwg-cp:hop>
      <nmwg-cp:hop id="3">
        <nmwg-cp:linkIdRef>
      urn:ogf:network:domain=domain2.net:node=1:port=1:link=1
        </nmwg-cp:linkIdRef>
      </nmwg-cp:hop>
      <nmwg-cp:hop id="4">
        <nmwg-cp:linkIdRef>
      urn:ogf:network:domain=domain2.net:node=2:port=1:link=1
        </nmwg-cp:linkIdRef>
      </nmwg-cp:hop>
      <nmwg-cp:hop id="5">
        <nmwg-cp:linkIdRef>
      urn:ogf:network:domain=domain3.net:node=1:port=1:link=1
        </nmwg-cp:linkIdRef>
      </nmwg-cp:hop>
      <nmwg-cp:hop id="6">
        <nmwg-cp:linkIdRef>
      urn:ogf:network:domain=domain3.net:node=2:port=1:link=1
        </nmwg-cp:linkIdRef>
      </nmwg-cp:hop>
    </idc:path>
    <idc:layer2Info>
      <idc:srcVtag tagged="true">3221</idc:srcVtag>
```

```
          <idc:destVtag tagged="true">3221</idc:destVtag>
          <idc:srcEndpoint>
          urn:ogf:network:domain=domain1.net:node=1:port=1:link=1
          </idc:srcEndpoint>
          <idc:destEndpoint>
          urn:ogf:network:domain=domain3.net:node=2:port=1:link=1
          </idc:destEndpoint>
        </idc:layer2Info>
      </idc:pathInfo>
     </idc:createReservation>
    </idc:payload>
   </idc:forward>
  </soap:Body>
</soap:Envelope>
```

Notice that the Domain 1 links are filled in and its listing potential VLANs that other domains can use in their resource scheduling decision. When Domain 2 receives this request it returns the following:

```
<soap:Envelope ...>
  <soap:Body …>
    <idc:forwardResponse>
     <idc:contentType>createReservation</idc:contentType>
     <idc:createReservationResponse>
      <idc:globalReservationId>domain1.net-
1</idc:globalReservationId>
      <idc:status>ACCEPTED</idc:status>
      <idc:pathInfo>
         [PATHINFO from Domain 1 repeated here]
      </idc:pathInfo>
    </idc:createReservationResponse>
   </idc:forwardResponse>
  </soap:Body>
</soap:Envelope>
```

After sending that reply Domain 2 calculates a path, fills in the domain 2 links in the path and forwards the request to Domain 3. Domain 3 also returns a reply simlar to above (NOTE: The *forward* and *forwardResponse* are not shown to save space). Finally Domain 3 selects the exact resources (in this case that includes VLAN) it will use and triggers a RESERVATION_CREATE_CONFIRMED event as shown below:

```
<soap:Envelope ...>
<soap:Body …>
 <wsnt:Notify>
```

```
  <wsnt:NotificationMessage>
    <wsnt:Topic Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Full">idc:IDC</ns5:Topic>
    <wsnt:ProducerReference>
      <wsa:Address>https://domain3.net/IDC</Address>
      <wsa:ReferenceParameters>
       <idc:subscriptionId>
         urn:uuid:097387e5-6b7c-4eec-a6c7-09a0466065e7
       </idc:subscriptionId>
     </wsaReferenceParameters>
   </wsnt:ProducerReference>
   <wsnt:Message>
    <idc:event id="event-1646550480">
      <idc:type>RESERVATION_CREATE_CONFIRMED</idc:type>
      <idc:timestamp>1210847836</idc:timestamp>
      <idc:userLogin>user1</idc:userLogin>
     <idc:resDetails>
      <idc:status>INCREATE</idc:status>
      <idc:login>user1</idc:login>
      <idc:globalReservationId>
       domain1.net-1
      </idc:globalReservationId>
      <idc:startTime>1210847896</idc:startTime>
      <idc:endTime>1213847896</idc:endTime>
      <idc:bandwidth>1000</idc:bandwidth>
      <idc:description>1 Gbps example</idc:description>
      <idc:pathInfo>
        <idc:pathSetupMode>timer-automatic<idc:pathSetupMode>
        <idc:path>
         <nmwg-cp:hop id=”1”>
          <nmwg-cp:link id=”
        urn:ogf:network:domain=domain1.net:node=1:port=1:link=1”>
           <trafficEngineeringMetric>
             10
           </trafficEngineeringMetric>
           <SwitchingCapabilityDescriptors>
             <switchingcapType>l2sc</switchingcapType>
             <encodingType>ethernet</encodingType>
             <switchingCapabilitySpecificInfo>
               <interfaceMTU>9000</interfaceMTU>
               <vlanRangeAvailability>
                3220-3224
               </vlanRangeAvailability>
```

```
            <suggestedVLANRange>3221</suggestedVLANRange>
          </switchingCapabilitySpecificInfo>
        </SwitchingCapabilityDescriptors>
      </nmwg-cp:link>
    </nmwg-cp:hop>
    <nmwg-cp:hop id="2">
      <nmwg-cp:link id="
urn:ogf:network:domain=domain1.net:node=2:port=1:link=1">
        <trafficEngineeringMetric>
          10
        </trafficEngineeringMetric>
        <SwitchingCapabilityDescriptors>
          <switchingcapType>l2sc</switchingcapType>
          <encodingType>ethernet</encodingType>
          <switchingCapabilitySpecificInfo>
            <interfaceMTU>9000</interfaceMTU>
            <vlanRangeAvailability>
             3220-3224
            </vlanRangeAvailability>
            <suggestedVLANRange>3221</suggestedVLANRange>
          </switchingCapabilitySpecificInfo>
        </SwitchingCapabilityDescriptors>
      </nmwg-cp:link>
    </nmwg-cp:hop>
    <nmwg-cp:hop id="3">
      <nmwg-cp:link id="
urn:ogf:network:domain=domain2.net:node=1:port=1:link=1">
        <trafficEngineeringMetric>
          10
        </trafficEngineeringMetric>
        <SwitchingCapabilityDescriptors>
          <switchingcapType>l2sc</switchingcapType>
          <encodingType>ethernet</encodingType>
          <switchingCapabilitySpecificInfo>
            <interfaceMTU>9000</interfaceMTU>
            <vlanRangeAvailability>
             3220-3224
            </vlanRangeAvailability>
            <suggestedVLANRange>3221</suggestedVLANRange>
          </switchingCapabilitySpecificInfo>
        </SwitchingCapabilityDescriptors>
      </nmwg-cp:link>
    </nmwg-cp:hop>
    <nmwg-cp:hop id="4">
      <nmwg-cp:link id="
urn:ogf:network:domain=domain2.net:node=2:port=1:link=1">
```

```
            <trafficEngineeringMetric>
              10
            </trafficEngineeringMetric>
            <SwitchingCapabilityDescriptors>
              <switchingcapType>l2sc</switchingcapType>
              <encodingType>ethernet</encodingType>
              <switchingCapabilitySpecificInfo>
                <interfaceMTU>9000</interfaceMTU>
                <vlanRangeAvailability>
                 3220-3224
                </vlanRangeAvailability>
                <suggestedVLANRange>3221</suggestedVLANRange>
              </switchingCapabilitySpecificInfo>
            </SwitchingCapabilityDescriptors>
          </nmwg-cp:link>
        </nmwg-cp:hop>
        <nmwg-cp:hop id="5">
          <nmwg-cp:link id="
      urn:ogf:network:domain=domain3.net:node=1:port=1:link=1">
            <trafficEngineeringMetric>
              10
            </trafficEngineeringMetric>
            <SwitchingCapabilityDescriptors>
              <switchingcapType>l2sc</switchingcapType>
              <encodingType>ethernet</encodingType>
              <switchingCapabilitySpecificInfo>
                <interfaceMTU>9000</interfaceMTU>
                <vlanRangeAvailability>
                 3221
                </vlanRangeAvailability>
              </switchingCapabilitySpecificInfo>
            </SwitchingCapabilityDescriptors>
          </nmwg-cp:link>
        </nmwg-cp:hop>
        <nmwg-cp:hop id="6">
          <nmwg-cp:link id="
      urn:ogf:network:domain=domain3.net:node=2:port=1:link=1">
            <trafficEngineeringMetric>
              10
            </trafficEngineeringMetric>
            <SwitchingCapabilityDescriptors>
              <switchingcapType>l2sc</switchingcapType>
              <encodingType>ethernet</encodingType>
              <switchingCapabilitySpecificInfo>
                <interfaceMTU>9000</interfaceMTU>
                <vlanRangeAvailability>
```

```
                3221
               </vlanRangeAvailability>
             </switchingCapabilitySpecificInfo>
           </SwitchingCapabilityDescriptors>
         </nmwg-cp:link>
       </nmwg-cp:hop>
      </idc:path>
      <idc:layer2Info>
        <idc:srcVtag tagged="true">3221</idc:srcVtag>
        <idc:destVtag tagged="true">3221</idc:destVtag>
        <idc:srcEndpoint>
        urn:ogf:network:domain=domain1.net:node=1:port=1:link=1
        </idc:srcEndpoint>
        <idc:destEndpoint>
        urn:ogf:network:domain=domain3.net:node=2:port=1:link=1
        </idc:destEndpoint>
      </idc:layer2Info>
    </idc:pathInfo>
   </idc:resDetails>
  </idc:event>
 </wsnt:Message>
</wsnt:NotificationMessage>
</wsnt:Notify>
</soap:Body>
</soap:Envelope>
```

Domain 2 receives this event, also finalizes the VLANS and passes a similar message back to Domain 1 (this message is not shown but the only different is that the VLANS for hops 3 and 4 are selected). Domain 1 finalizes it resources and throws the following RESERVATION_CREATE_COMPLETED event to indicate the reservation was reserved:

```
<soap:Envelope ...>
<soap:Body …>
 <wsnt:Notify>
  <wsnt:NotificationMessage>
    <wsnt:Topic Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Full">idc:IDC</ns5:Topic>
    <wsnt:ProducerReference>
      <wsa:Address>https://domain3.net/IDC</Address>
      <wsa:ReferenceParameters>
       <idc:subscriptionId>
         urn:uuid:097387e5-6b7c-4eec-a6c7-09a0466065e7
       </idc:subscriptionId>
```

```
    </wsaReferenceParameters>
  </wsnt:ProducerReference>
  <wsnt:Message>
   <idc:event id="event-1646550480">
     <idc:type>RESERVATION_CREATE_COMPLETED</idc:type>
     <idc:timestamp>1210847836</idc:timestamp>
     <idc:userLogin>user1</idc:userLogin>
    <idc:resDetails>
     <idc:status>PENDING</idc:status>
     <idc:login>user1</idc:login>
     <idc:globalReservationId>
      domain1.net-1
     </idc:globalReservationId>
     <idc:startTime>1210847896</idc:startTime>
     <idc:endTime>1213847896</idc:endTime>
     <idc:bandwidth>1000</idc:bandwidth>
     <idc:description>1 Gbps example</idc:description>
     <idc:pathInfo>
       <idc:pathSetupMode>timer-automatic<idc:pathSetupMode>
       <idc:path>
        <nmwg-cp:hop id="1">
         <nmwg-cp:link id="
       urn:ogf:network:domain=domain1.net:node=1:port=1:link=1">
          <trafficEngineeringMetric>
            10
          </trafficEngineeringMetric>
          <SwitchingCapabilityDescriptors>
            <switchingcapType>l2sc</switchingcapType>
            <encodingType>ethernet</encodingType>
            <switchingCapabilitySpecificInfo>
              <interfaceMTU>9000</interfaceMTU>
              <vlanRangeAvailability>
               3220-3224
              </vlanRangeAvailability>
              <suggestedVLANRange>3221</suggestedVLANRange>
            </switchingCapabilitySpecificInfo>
          </SwitchingCapabilityDescriptors>
         </nmwg-cp:link>
        </nmwg-cp:hop>
        <nmwg-cp:hop id="2">
          <nmwg-cp:link id="
       urn:ogf:network:domain=domain1.net:node=2:port=1:link=1">
          <trafficEngineeringMetric>
            10
```

```
            </trafficEngineeringMetric>
            <SwitchingCapabilityDescriptors>
              <switchingcapType>l2sc</switchingcapType>
              <encodingType>ethernet</encodingType>
              <switchingCapabilitySpecificInfo>
                <interfaceMTU>9000</interfaceMTU>
                <vlanRangeAvailability>
                 3220-3224
                </vlanRangeAvailability>
                <suggestedVLANRange>3221</suggestedVLANRange>
              </switchingCapabilitySpecificInfo>
            </SwitchingCapabilityDescriptors>
          </nmwg-cp:link>
        </nmwg-cp:hop>
        <nmwg-cp:hop id="3">
         <nmwg-cp:link id="
      urn:ogf:network:domain=domain2.net:node=1:port=1:link=1">
            <trafficEngineeringMetric>
              10
            </trafficEngineeringMetric>
            <SwitchingCapabilityDescriptors>
              <switchingcapType>l2sc</switchingcapType>
              <encodingType>ethernet</encodingType>
              <switchingCapabilitySpecificInfo>
                <interfaceMTU>9000</interfaceMTU>
                <vlanRangeAvailability>
                 3221
                </vlanRangeAvailability>
              </switchingCapabilitySpecificInfo>
            </SwitchingCapabilityDescriptors>
          </nmwg-cp:link>
        </nmwg-cp:hop>
        <nmwg-cp:hop id="4">
         <nmwg-cp:link id="
      urn:ogf:network:domain=domain2.net:node=2:port=1:link=1">
            <trafficEngineeringMetric>
              10
            </trafficEngineeringMetric>
            <SwitchingCapabilityDescriptors>
              <switchingcapType>l2sc</switchingcapType>
              <encodingType>ethernet</encodingType>
              <switchingCapabilitySpecificInfo>
                <interfaceMTU>9000</interfaceMTU>
                <vlanRangeAvailability>
                 3221
                </vlanRangeAvailability>
```

```
            </switchingCapabilitySpecificInfo>
          </SwitchingCapabilityDescriptors>
        </nmwg-cp:link>
      </nmwg-cp:hop>
      <nmwg-cp:hop id="5">
        <nmwg-cp:link id="
    urn:ogf:network:domain=domain3.net:node=1:port=1:link=1">
          <trafficEngineeringMetric>
            10
          </trafficEngineeringMetric>
          <SwitchingCapabilityDescriptors>
            <switchingcapType>l2sc</switchingcapType>
            <encodingType>ethernet</encodingType>
            <switchingCapabilitySpecificInfo>
              <interfaceMTU>9000</interfaceMTU>
              <vlanRangeAvailability>
                3221
              </vlanRangeAvailability>
            </switchingCapabilitySpecificInfo>
          </SwitchingCapabilityDescriptors>
        </nmwg-cp:link>
      </nmwg-cp:hop>
      <nmwg-cp:hop id="6">
        <nmwg-cp:link id="
    urn:ogf:network:domain=domain3.net:node=2:port=1:link=1">
          <trafficEngineeringMetric>
            10
          </trafficEngineeringMetric>
          <SwitchingCapabilityDescriptors>
            <switchingcapType>l2sc</switchingcapType>
            <encodingType>ethernet</encodingType>
            <switchingCapabilitySpecificInfo>
              <interfaceMTU>9000</interfaceMTU>
              <vlanRangeAvailability>
                3221
              </vlanRangeAvailability>
            </switchingCapabilitySpecificInfo>
          </SwitchingCapabilityDescriptors>
        </nmwg-cp:link>
      </nmwg-cp:hop>
    </idc:path>
    <idc:layer2Info>
      <idc:srcVtag tagged="true">3221</idc:srcVtag>
      <idc:destVtag tagged="true">3221</idc:destVtag>
      <idc:srcEndpoint>
      urn:ogf:network:domain=domain1.net:node=1:port=1:link=1
```

```
          </idc:srcEndpoint>
          <idc:destEndpoint>
          urn:ogf:network:domain=domain3.net:node=2:port=1:link=1
          </idc:destEndpoint>
        </idc:layer2Info>
      </idc:pathInfo>
     </idc:resDetails>
    </idc:event>
   </wsnt:Message>
  </wsnt:NotificationMessage>
 </wsnt:Notify>
</soap:Body>
</soap:Envelope>
```

The event above is passed to both the original requester and Domain 2. Domain 2 passes the message exactly to Domain 3. Once completed the resources are held in every domain.

# 18 References


**[NMWG-CP]** Network Measurement Working Group (NMWG) Control Plane Schemas. Based on Revision 378 of NMWG Schemas.

**[CNTL-PLANE]** The DICE IDCP Control Plane Web Site, http://www.controlplane.net

**[DigSig]** XML-Signature Syntax and Processing: D. Eastlake 3rd,J. Reagle, D. Solo, RFC3275 Sept 2002. http://www.ietf.org/rfc/rfc3275.txt

**[RFC2119]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997. http://www.ietf.org/rfc/rfc2119.txt

**[SOAP]** "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation. http://www.w3.org/TR/soap12-part1/

**[WSDL]** "Web Services Description Language (WSDL) 1.1", W3C Note. http://www.w3.org/TR/wsdl

**[WSBN]** http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf

**[WSN]** http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

**[WSA]** http://www.w3.org/2005/08/addressing

**[WS-Sec]** "Web Services Security SOAP Message Security 1.1 (WS-Security 2004)", OASIS Standard Specification, 1 February 2006. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

**[XML-Infoset]** "XML Information Set", W3C Recommendation. http://www.w3.org/TR/xml-infoset/

**[XPath]** "XML Path Language (XPath) Version 1.0", W3C Recommendation. http://www.w3.org/TR/xpath

**[XML Schema]** W3C Recommendation, "XML Schema Part 1: Structures,"2 May 2001. http://www.w3.org/TR/xmlschema11-1/
W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001. http://www.w3.org/TR/xmlschema11-2/

**[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998. http://www.ietf.org/rfc/rfc2396.txt

**[URN]** R. Moats, "URN Syntax", RFC 2141, AT&T, May 1997. http://www.ietf.org/rfc/rfc2141.txt