Inter-domain Controller (IDC) Protocol Specification

Andrew Lake, John Vollbrecht, Aaron Brown, Jason Zurawski: Internet2;
David Robertson, Mary Thompson, Chin Guok, , Evangelos Chaniotakis: ESnet
Tom Lehman: ISI-East

May 30, 2008

Table of Contents

1 Introduction

This document specifies the Inter-domain Controller (IDC) protocol for dynamically provisioning network resources across multiple administrative domains. Specifically, the IDC protocol is designed to support services implementing the architecture described in the IDC Architecture document [IDC-Arch]. The architecture describes *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, gathering information about previously requested resources, and basic topology exchange. These messages are defined in a SOAP [SOAP] web service format. Since all messages are defined using SOAP, the protocol also utilizes a few external web service protocols and XML descriptions for features such as security and topology description. Later sections in this document will indicate where external protocols 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.

1.1 Goals and Requirements

The goal of the IDC protocol is to standardize the terminology, concepts, operations, WSDL and XML 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 monitoring and measurement. The IDC protocol should utilize external protocols and XML definitions 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 [IDC-Arch] 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.

- **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 the 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 [IDC-Arch] 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 a service.

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 to all domains and often formed by appending a locally unique number to the globally unique domain identifier of the IDC receiving the 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. 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
nmwg-cp	http://ogf.org/schema/network/topology/ctrlPlane/20070626/
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
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 Models

The Inter-domain Controller architecture [IDC-Arch] defines two messaging models: the daisy chain model and the meta-scheduler model. The messaging model implemented determines how IDCs are required to interact. The protocol defined in this document provides mechanisms that support both daisy-chaining and meta-scheduling. These mechanisms are explored further in this section.

2.1 Daisy-Chain Messaging

The daisy-chain model 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. *Figure 2.1* shows an example of a daisy chain between three domains.

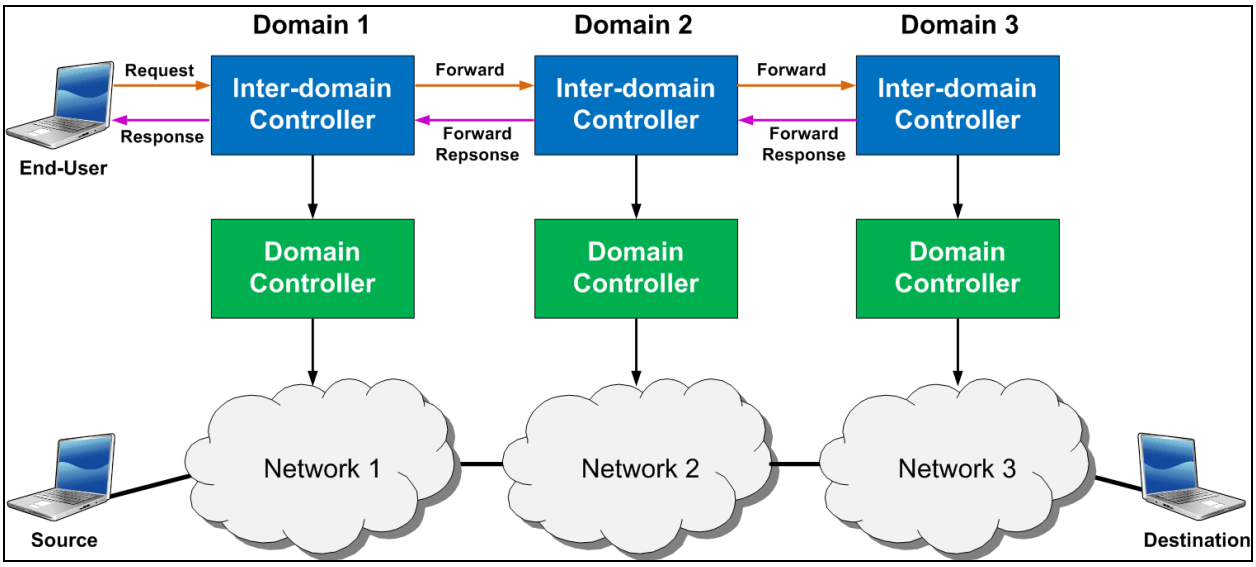

Figure 2.1 *An example of a daisy-chain model with three domains.*

In the figure 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. The IDC of Domain 1 modifies the request and encapsulates it in a <idc:forward> message to the next domain. Likewise, Domain 2 extracts the request parameters from the <idc:forward> message, modifies it further if necessary, and then encapsulates it in a message to Domain 3. Domain 3 processes the request and then returns a <idc:forwardResponse> message back to Domain 2. Domain 2 also sends an <idc:forwardResponse> to Domain 1 which then responds to the end-user's original request. The format of the messages sent between end-user and the initial IDC as well as the <idc:forward> and <idc:forwardResponse> message sent between IDCs is described in the remainder of this section.

2.1.1 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 **3** of this document. The SOAP body of

messages may be one of the several types defined in sections Error! Reference source not found. thru Error! Reference source not found.. 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 **2.1.2**).

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="refreshPath">
    <wsdl:input message="tns:refreshPath" />
    <wsdl:output message="tns:refreshPathResponse" />
    <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 Error! Reference source not found. thru Error! Reference source not found. of this document.

2.1.2 IDC to IDC Message Forwarding

Messages are passed between IDCs along a daisy chain using the *forward* operation. 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 Error! Reference source not found. thru Error! Reference source not found..

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 Error! Reference source not found. thru Error! Reference source not found. of this document.

2.2 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 2.2* shows a diagram describing the meta-scheduler model:

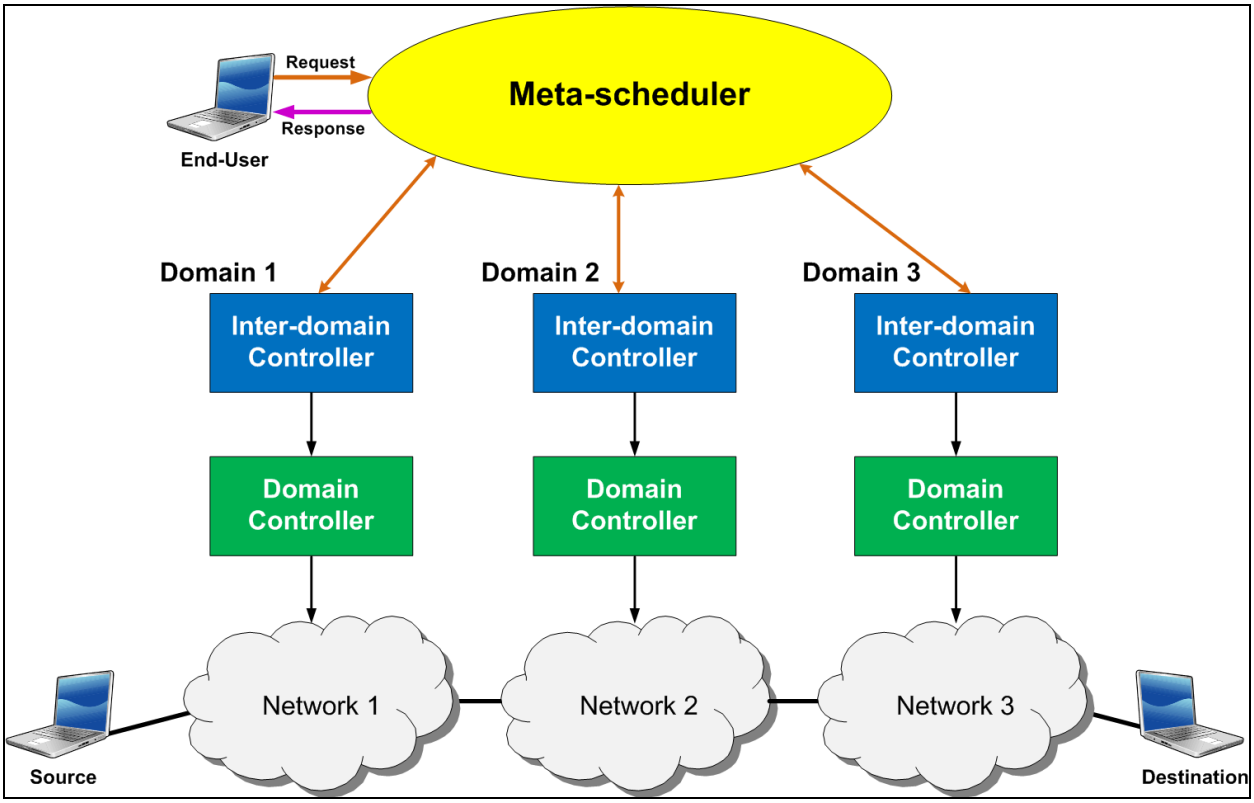

Figure 2.2 *An example of the meta-scheduler model with 3 domains*

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 **2.1.1** 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.

3 Security

3.1 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 time stamp, 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.

3.2 Digital Signature Format and Algorithms

In theory our 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.

3.3 Example

3.3.1 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>
```

3.3.2 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>
```

4 Common Data Types

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

4.1 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, and can be one of ACTIVE, PENDING, FINISHED, CANCELLED, and FAILED. ACTIVE indicates the reservation has been scheduled and the circuit is currently reserved, PENDING indicates the reservation has been scheduled but the circuit has not be reserved yet, and FINISHED indicates the circuit completed its lifetime normally and the reservation is no longer active. CANCELLED indicates that the user cancelled the reservation and the circuit was torn down before its scheduled time, and FAILED indicates that the reservation could not be scheduled. More on the process of resource scheduling is given in section 5.

/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.

4.2 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 **6.1**) or have the user initiate circuit setup with the *createPath* message (see section **6.2.1)**.

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 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: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.

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 port number at the source.

idc:layer3Info/idc:destIpPort

This field MAY be included, and is the 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 internally:

```
<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

5 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.

5.1 Creating a Reservation

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 **4.1**); all technology-specific information is contained in the pathInfo sub-object (see section **4.2**). 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 **4.2**. The path information here represents what the reservation requester is asking from the IDC.

The response of a *createReservation* operation is described 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 be included. Contains the current reservation status, which will typically be PENDING since the reservation has just been created.

/idc:createReservationResponse/idc:pathInfo

MUST be included, and is extensively described in section **4.2**. This path information describes the path the IDC decided that the reservation will actually take.

5.1.1 Path Calculation

TODO: COMPLETE THIS SECTION

5.1.2 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 2.1*. 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 then the request is passed to Domain 2 and Domain 3 according the daisy chain model. Each IDC expands and verifies the reservation path and when complete Domain 1 returns a successful response as shown below:

```
<soap:Envelope ...>
  <soap:Body …>
    <idc:createReservationResponse>
      <idc:globalReservationId>domain1.net-
1</idc:globalReservationId>
      <idc:status>PENDING</idc:status>
      <idc:pathInfo>
        <idc:pathSetupMode>timer-automatic<idc:pathSetupMode>
        <idc:path>
         <nmwg-cp:hop id=”1”>
           <nmwg-cp:linkIdRef>
         urn:ogf:network:domain=domain1.net:node=1:port=1:link=1
           </nmwg-cp:linkIdRef>
         </nmwg-cp:hop>
         <nmwg-cp:hop id=”2”>
           <nmwg-cp:linkIdRef>
         urn:ogf:network:domain=domain1.net:node=2:port=1:link=1
           </nmwg-cp:linkIdRef>
         </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”>2988</idc:srcVtag>
          <idc:destVtag tagged=”true”>2988</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:createReservationResponse>
  </soap:Body>
</soap:Envelope>
```

Note that the IDC has generated a GRI, has converted the hostnames provided by the user to URNs through the Lookup Service, has negotiated an inter-domain path and a VLAN number, and provides all this information back to the user in the response.

5.2 Modifying a Reservation

This message is used to modify an existing reservation. 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 **4.2**. The path information here will replace the existing path information.

The *modifyReservation* response message is shown below:

```
<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 **4.1**.

5.3 Cancelling a Reservation

Cancellation of a reservation is used to correct erroneous reservations, or to terminate active reservations before the originally defined end time. When receiving a valid cancellation message the IDC will immediately release any resources held by the reservation. Additionally, if the reservation was active that path will immediately be torn down as well.

The request message 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.

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 successful”.

6 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.

6.1 Automatic 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 **7.2**). End-users or IDCs wishing to have more direct control over a circuit may want to consider using message signaling.

6.2 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. It is also useful between IDCs because it provides some indication of circuit status in downstream domains. For this reason it is RECOMMENDED that IDCs use message signaling between other IDCs[1].

Message signaling can be specified between IDCs no matter what the end-user requests by always setting <idc:pathSetupMode> to *signal-xml* in the initial IDC when forwarding resource scheduling messages. In such as case, the initial IDC MUST return *timer-automatic* to the end-user and automatically send signaling messages to the first downstream IDC at the start and end time of the reservation (i.e. the user does NOT have to send any signaling messages). The signaling messages for creating, refreshing and tearing down a circuit are detailed in the remainder of this section.

6.2.1 Creating a Circuit

After a reservation has been placed requiring message signaling (see section **5**) 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="token" type="xsd:string" minOccurs="0"
      maxOccurs="1"/>
    <xsd:element name="globalReservationId" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

/idc:createPath

[1] This recommendation may change in the future if a more complex IDC-to-IDC notification scheme is devised

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 **2.1.1**). If this element is passing between IDCs it will be encapsulated in an <idc:forward> element (see section **2.1.2**).

/idc:createPath/idc:token

An optional hex string representing an authorization token generated during resource scheduling. Some tokens MAY be transferable between end-users. If the sender of this message is different from the entity that placed the reservation then a transferable token is required. If the sender is the same as the entity that place the reservations then an IDC MAY NOT require a token if necessary authorization information can be derived from the message security headers (see section **3**).

/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 **2.1.1**). If this element is passing between IDCs it will be encapsulated in an <idc:forwardResponse> element (see section **2.1.2**).

/idc:createPathResponse /idc:status

The status that resulted from the operation. It should have a value of ACTIVE if the circuit was successfully created.

/idc:createPathResponse /idc:globalReservationId

A required field indicating the global reservation identifier (GRI) of the reservation with the circuit that was created.

An IDC SHOULD 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 verify that the requester is authorized to signal the reservation.
2. Upon authorization the IDC should immediately 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.
4. Upon successful creation of the circuit by the DC, the IDC should return a response to the requester indicating the circuit has been created.

In step 2, forwarding this request prior to circuit creation is recommended because it allows the local IDC to obtain the status of downstream domains before building its portion of the dynamic circuit. If circuit creation in step 3 fails an IDC MAY send a <idc:teardownPath> or <idc:cancelReservation> message to the next IDC in the reservation's path and MUST return a fault to the requester.

6.2.2 Refreshing a Circuit

An IDC MAY require periodic keep-alive messages for a circuit to remain active. It may also want to check the status of the data plane in the local domain to make sure that no errors have occurred with the circuit. Processing keep-alive messages and verifying the data plane is often referred to as "refreshing" a circuit. This specification defines a *refreshPath* operation that triggers this function. If an error is detected in the data plane while performing the refresh an IDC MAY send a *cancelReservation* OR *teardownPath* message to neighboring IDCs. Many domains MAY NOT implement the *refreshPath* operation as its use has yet to be well-defined. Future versions of this specification may expand upon to use of the *refreshPath* operation. The *refreshPath* request is described below:

```
<xsd:element name="refreshPath" type="tns:refreshPathContent" />
<xsd:complexType name="refreshPathContent">
  <xsd:sequence>
```

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

/idc:refreshPath

A container element with parameters for refreshing 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 **2.1.1**). If this element is passing between IDCs it will be encapsulated in an <idc:forward> element (see section **2.1.2**).

/idc:refreshPath/idc:token

An optional hex string representing an authorization token generated during resource scheduling. Some tokens MAY be transferable between end-users. If the sender of this message is different from the entity that placed the reservation then a transferable token is required. If the sender is the same as the entity that place the reservations then an IDC MAY NOT require a token if necessary authorization information can be derived from the message security headers (see section **3**).

/idc:refreshPath/idc:globalReservationId

A required field indicating the global reservation identifier (GRI) of the reservation with the circuit to refresh.

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

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

/idc:refreshPathResponse

A container element returning the results of the *refreshPath* 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 **2.1.1**). If this element is passing between

IDCs it will be encapsulated in an <idc:forwardResponse> element (see section **2.1.2**).

/idc:refreshPathResponse/idc:status

The status that resulted from the operation. It should have a value of ACTIVE if the circuit still exists in the data plane. If an error occurred in the data plane a fault should be thrown.

/idc:refreshPathResponse/idc:globalReservationId

A required field indicating the global reservation identifier (GRI) of the reservation that was refreshed.

6.2.3 Tearing down a Circuit

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 **5.3**) 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="token" type="xsd:string" minOccurs="0"
      maxOccurs="1"/>
    <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 **2.1.1**). If this element is passing between IDCs it will be encapsulated in an <idc:forward> element (see section **2.1.2**).

/idc:teardownPath/idc:token

An optional hex string representing an authorization token generated during resource scheduling. Some tokens MAY be transferable between end-users. If

the sender of this message is different from the entity that placed the reservation then a transferable token is required. If the sender is the same as the entity that place the reservations then an IDC MAY NOT require a token if necessary authorization information can be derived from the message security headers (see section **3**).

/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 **2.1.1**). If this element is passing between IDCs it will be encapsulated in an <idc:forwardResponse> element (see section **2.1.2**).

/idc:teardownPathResponse/idc:status

The status that resulted from the operation. It should have a value of PENDING if the circuit was successfully removed.

/idc:teardownPathResponse/idc:globalReservationId

A required field indicating the global reservation identifier (GRI) of the circuit that was removed.

6.2.4 Examples

The section provides an example of signaling messages used to create and teardown a circuit belonging to a reservation with global reservation identifier (GRI) *domain1.net - 1*.It assumes three domains are in the reservation path as shown in *Figure 2.1*. Domain 1 will be the initial domain and it will locally identify the user with string *enduser1*.

Once the start time of reservation *domain1.net -1* is reached an end-user may send a *createPath* request as follows:

```
<soap:Envelope ...>
  <soap:Header>
    [End-user security credentials]
  </soap:Header>
  <soap:Body ...>
    <idc:createPath ...>
     <idc:globalReservationId>domain1.net-
1<idc:globalReservationId>
    </idc:createPath>
  </soap:Body>
</soap:Envelope>
```

The IDC of Domain 1 shown in *Figure 2.1* will receive this message and authorize it. Since no token is provided the same end-user that made the reservation must also be sending this request. Assuming that this is the case, Domain 1 forwards the following to Domain 2's IDC:

```
<soap:Envelope ...>
  <soap:Header>
    [Domain 1 security credentials]
  </soap:Header>
  <soap:Body ...>
   <idc:forward ...>
    <idc:payload>
     <idc:contentType>createPath</idc:contentType>
     <idc:createPath ...>
      <idc:globalReservationId>
      domain1.net-1<idc:globalReservationId>
     </idc:createPath>
    </idc:payload>
    <idc:payloadSender>enduser1</idc:payloadSender>
   </idc:forward>
  </soap:Body>
</soap:Envelope>
```

Domain 2 will also authorize this request and then immediately forward the same request to Domain 3 (the only difference in the request will be that the security header will contain credentials for Domain 2 rather than Domain 1). Since Domain 3 is the last

domain in the path, instead of forwarding the message it will contact its domain controller (DC) with instructions to create Network 3's portion of the circuit. When complete it will return the following to Domain 2:

```
<soap:Envelope ...>
  <soap:Header>
    …
  </soap:Header>
  <soap:Body ...>
   <idc:forwardResponse …>
    <idc:contentType>createPathResponse</idc:contentType>
    <idc:createPathResponse ...>
     <idc:globalReservationId>domain1.net-
1<idc:globalReservationId>
     <idc:status>ACTIVE</idc:status>
    </idc:createPathResponse >
   </idc:forwardResponse>
  </soap:Body>
</soap:Envelope>
```

Domain 2 will then contact its DC to create Network 2's portion of the circuit. It will return a response to Domain 1 that looks the same as that which Domain 2 received from Domain 3. Domain 1's IDC will then contact its DC to create the local portion of the circuit on Network 1. This results in the entire end-to-end circuit being created. It will then return the following response to the end-user to complete the operation:

```
<soap:Envelope ...>
  <soap:Header>
    …
  </soap:Header>
  <soap:Body ...>
    <idc:createPathResponse ...>
     <idc:globalReservationId>domain1.net-
1<idc:globalReservationId>
     <idc:status>ACTIVE</idc:status>
    </idc:createPathResponse >
  </soap:Body>
</soap:Envelope>
```

Once the end-user receives the above response the circuit can be used to transmit data between the endpoints.

When the circuit is no longer needed the end-user can either wait for the circuit to expire or send a *teardownPath* before the reservation ends. Assuming the end-user chooses the latter option, below is an example of the *teardownPath* request sent by the end-user to Domain 1's IDC:

```
<soap:Envelope ...>
  <soap:Header>
    [End-user security credentials]
  </soap:Header>
  <soap:Body ...>
    <idc:teardownPath ...>
     <idc:globalReservationId>domain1.net-
1<idc:globalReservationId>
    </idc:teardownPath>
  </soap:Body>
</soap:Envelope>
```

Domain 1's IDC will authorize the request, request that the DC teardown the circuit, and forward the request to Domain 2. Below is an example of the request sent to Domain 2's IDC:

```
<soap:Envelope ...>
  <soap:Header>
    [Domain 1 security credentials]
  </soap:Header>
  <soap:Body ...>
   <idc:forward ...>
    <idc:payload>
     <idc:contentType>teardownPath</idc:contentType>
     <idc:teardownPath ...>
      <idc:globalReservationId>
      domain1.net-1<idc:globalReservationId>
     </idc:teardownPath>
    </idc:payload>
    <idc:payloadSender>enduser1</idc:payloadSender>
   </idc:forward>
  </soap:Body>
</soap:Envelope>
```

Domain 2 will authorize the request, teardown the path, and forward the request to Domain 3 (the request is the same as above, except with Domain 2's authorization

credentials). Domain 3 will do the same and since it is the last domain in the path the entire circuit is removed from the data plane. Domain 3's IDC will return the following to Domain 2:

```
<soap:Envelope ...>
  <soap:Header>
    …
  </soap:Header>
  <soap:Body ...>
   <idc:forwardResponse …>
    <idc:contentType>teardownPathResponse</idc:contentType>
    <idc:teardownPathResponse ...>
     <idc:globalReservationId>domain1.net-
1<idc:globalReservationId>
     <idc:status>PENDING</idc:status>
    </idc:teardownPathResponse >
   </idc:forwardResponse>
  </soap:Body>
</soap:Envelope>
```

Domain 2 will return a similar response to Domain 1. Domain 1 will then return the following to the end-user to complete the operation:

```
<soap:Envelope ...>
  <soap:Header>
    …
  </soap:Header>
  <soap:Body ...>
    <idc:teardownPathResponse ...>
     <idc:globalReservationId>domain1.net-
1<idc:globalReservationId>
     <idc:status>PENDING</idc:status>
    </idc:teardownPathResponse >
  </soap:Body>
</soap:Envelope>
```

7 Monitoring

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).

7.1 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.

7.1.1 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>
```

7.2 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.

7.2.1 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.

8 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. 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.

9 References

[IDC-Arch] TBD

[NMWG-CP] TBD

[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

[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