Web Services Dynamic Discovery (WS-Discovery)

April 2005

Co-Developers

John Beatty, BEA Systems
Gopal Kakivaya, Microsoft
Devon Kemp, Canon
Thomas Kuehnel, Microsoft
Brad Lovering, Microsoft
Bryan Roe, Intel
Christopher St. John, webMethods
Jeffrey Schlimmer (Editor), Microsoft
Guillaume Simonnet, Microsoft
Doug Walter, Microsoft
Jack Weast, Intel
Yevgeniy Yarmosh, Intel
Prasad Yendluri, webMethods

Copyright Notice

Abstract

This specification defines a multicast discovery protocol to locate services. By default, probes are sent to a multicast group, and target services that match return a response directly to the requester. To scale to a large number of endpoints, the protocol defines the multicast suppression behavior if a discovery proxy is available on the network. To minimize the need for polling, target services that wish to be discovered send an announcement when they join and leave the network.

Composable Architecture

The Web service specifications (WS-*) are designed to be composed with each other to provide a rich set of tools to provide security in the Web services environment. This specification specifically relies on other Web service specifications to provide secure, reliable, and/or transacted message delivery and to express Web service and client policy.

Status

WS-Discovery and related specifications are provided for use as-is and for review and evaluation only. Microsoft, BEA, Canon, Intel, and webMethods will solicit your contributions and suggestions in the near future. Microsoft, BEA, Canon, Intel, and webMethods make no warrantees or representations regarding the specification in any manner whatsoever.

Table of Contents

1. Introduction

This specification defines a multicast discovery protocol to locate services. The primary mode of discovery is a client searching for one or more target services. To find a target service by the type of the target service, a scope in which the target service resides, or both, a client sends a probe message to a multicast group; target services that match the probe send a response directly to the client. To locate a target service by name, a client sends a resolution request message to the same multicast group, and again, the target service that matches sends a response directly to the client.

To minimize the need for polling, when a target service joins the network, it sends an announcement message to the same multicast group. By listening to this multicast group, clients can detect newly-available target services without repeated probing.

To scale to a large number of endpoints, this specification defines multicast suppression behavior if a discovery proxy is available on the network. Specifically, when a discovery proxy detects a probe or resolution request sent by multicast, the discovery proxy sends an announcement for itself. By listening for these announcements, clients detect discovery proxies and switch to use a discovery proxy-specific protocol. However, if a discovery proxy is unresponsive, clients revert to use the protocol described herein.

To support networks with explicit network management services like DHCP, DNS, domain controllers, directories, etc., this specification acknowledges that clients and/or target services may be configured to behave differently than defined herein. For example, another specification may define a well-known DHCP record containing the address of a discovery proxy, and compliance with that specification may require endpoints to send messages to this discovery proxy rather than to a multicast group. While the specific means of such configuration is beyond the scope of this specification, it is expected that any such configuration would allow clients and/or

target services to migrate smoothly between carefully-managed and ad hoc networks.

1.1 Requirements

This specification intends to meet the following requirements:

- Allow discovery of services in ad hoc networks with a minimum of networking services (e.g., no DNS or directory services).
- Leverage network services to reduce network traffic in managed networks where such services exist.
- Enable smooth transitions between ad hoc and managed networks.
- Enable discovery of resource-limited service implementations.
- Support bootstrapping to other Web service protocols as well as other transports.
- Enable discovery of services by type and within scope.
- Leverage other Web service specifications for secure, reliable, transacted message delivery.
- Provide extensibility for more sophisticated and/or currently unanticipated scenarios.
- Support both SOAP 1.1 [SOAP 1.1] and SOAP 1.2 [SOAP 1.2] Envelopes.

1.2 Non-Requirements

This specification does not intend to meet the following requirements:

- Provide liveness information on services.
- Define a data model for service description or define rich queries over that description.
- Support Internet-scale discovery.

1.3 Example

Table 1 lists an example Probe message multicast by a Client searching for a printer.

Table 1: Example Probe.

```
(01) <s:Envelope
(02)     xmlns:a="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(03)     xmlns:d="http://schemas.xmlsoap.org/ws/2005/04/discovery"
(04)     xmlns:i="http://printer.example.org/2003/imaging"
(05)     xmlns:s="http://www.w3.org/2003/05/soap-envelope" >
(06)   <s:Header>
(07)     <a:Action>
(08)       http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe
(09)     </a:Action>
(10)     <a:MessageID>
(11)       uuid:0a6dc791-2be6-4991-9af1-454778a1917a
(12)     </a:MessageID>
(13)     <a:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</a:To>
(14)   </s:Header>
(15)   <s:Body>
```

```
(16)     <d:Probe>
(17)       <d:Types>i:PrintBasic</d:Types>
(18)       <d:Scopes
(19)    MatchBy="http://schemas.xmlsoap.org/ws/2005/04/discovery/ldap" >
(20)         ldap:///ou=engineering,o=examplecom,c=us
(21)       </d:Scopes>
(22)     </d:Probe>
(23)   </s:Body>
(24) </s:Envelope>
(25)
```

Lines (07-09) in Table 1 indicate the message is a Probe, and Line (13) indicates it is being sent to a well-known address [RFC 2141].

Because there is no explicit ReplyTo SOAP header block [WS-Addressing], any response to this Probe will be sent as a UDP packet to the source IP address and port of the Probe transport header [SOAP/UDP].

Lines (17-21) specify two constraints on the Probe: Line (17) constrains responses to Target Services that implement a basic print Type; Lines (18-21) constrain responses to Target Services in the Scope for an engineering department. Only Target Services that satisfy both of these constraints will respond. Though both constraints are included in this example, a Probe is not required to include either.

Table 2 lists an example Probe Match message sent in response to the Probe in Table 1.

Table 2: Example Probe Match.

```
(01) <s:Envelope
(02)   xmlns:a="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(03)   xmlns:d="http://schemas.xmlsoap.org/ws/2005/04/discovery"
(04)   xmlns:i="http://printer.example.org/2003/imaging"
(05)   xmlns:s="http://www.w3.org/2003/05/soap-envelope" >
(06)   <s:Header>
(07)     <a:Action>
(08)       http://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches
(09)     </a:Action>
(10)     <a:MessageID>
(11)       uuid:e32e6863-ea5e-4ee4-997e-69539d1ff2cc
(12)     </a:MessageID>
(13)     <a:RelatesTo>
(14)       uuid:0a6dc791-2be6-4991-9af1-454778a1917a
(15)     </a:RelatesTo>
(16)     <a:To>
(17)     http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(18)     </a:To>
(19)     <d:AppSequence InstanceId="1077004800" MessageNumber="2" />
(20)   </s:Header>
```

```
(21)   <s:Body>
(22)     <d:ProbeMatches>
(23)       <d:ProbeMatch>
(24)         <a:EndpointReference>
(25)           <a:Address>
(26)             uuid:98190dc2-0890-4ef8-ac9a-5940995e6119
(27)           </a:Address>
(28)         </a:EndpointReference>
(29)         <d:Types>i:PrintBasic i:PrintAdvanced</d:Types>
(30)         <d:Scopes>
(31)           ldap:///ou=engineering,o=examplecom,c=us
(32)           ldap:///ou=floor1,ou=b42,ou=anytown,o=examplecom,c=us
(33)           http://itdept/imaging/deployment/2004-12-04
(34)         </d:Scopes>
(35)         <d:XAddrs>http://prn-example/PRN42/b42-1668-a</d:XAddrs>
(36)         <d:MetadataVersion>75965</d:MetadataVersion>
(37)       </d:ProbeMatch>
(38)     </d:ProbeMatches>
(39)   </s:Body>
(40) </s:Envelope>
(41)
```

Lines (07-09) in Table 2 indicate this message is a Probe Match, and Lines (13-15) indicate that it is a response to the Probe in Table 1. Because the Probe did not have an explicit ReplyTo SOAP header block, Lines (16-18) indicate that the response was sent to the source IP address and port of the transport header of the Probe. Line (19) contains an instance identifier as well as a message number; this information allows the receiver to reorder discovery messages received from a Target Service.

Lines (23-37) describe a single Target Service.

Lines (24-28) contain the stable, unique identifier for the Target Service that is constant across network interfaces, transport addresses, and IPv4/v6. In this case, the value is a UUID scheme URI, but it may be a transport URI (like the one in Line 35) if it meets stability and uniqueness requirements.

Line (29) lists the Types (see, e.g., [WSDL 1.1]) implemented by the Target Service, in this example, a basic print type that matched the Probe as well as an advanced print type.

Lines (30-34) list three administrative Scopes, one that matched the Probe (Line 31), one that is specific to a particular physical location (Line 32), and one that includes data useful when switching over to new infrastructure (Line 33). As in this case, the Scopes may be a heterogeneous collection of deployment-related information.

Line (35) indicates the transport addresses where the Target Service may be reached; in this case, a single HTTP transport address.

Line (36) contains the version of the metadata for the Target Service; as explained below, this version is incremented if there is a change in the metadata for the Target Service (including Lines 29-34).

2. Terminology and Notation

2.1 Terminology

Target Service
: An endpoint that makes itself available for discovery.

Client
: An endpoint that searches for Target Service(s).

Discovery Proxy
: An endpoint that facilitates discovery of Target Services by Clients. Discovery Proxies are an optional component of the architecture.

Hello
: A message sent by a Target Service when it joins a network; this message contains key information for the Target Service.

Bye
: A best-effort message sent by a Target Service when it leaves a network.

Probe
: A message sent by a Client searching for a Target Service by Type and/or Scope.

Resolve
: A message sent by a Client searching for a Target Service by name.

Type
: An identifier for a set of messages an endpoint sends and/or receives (e.g., a WSDL 1.1 portType, see [WSDL 1.1]).

Scope
: An extensibility point that may be used to organize Target Services into logical groups.

Metadata
: Information about the Target Service; includes, but is not limited to, transports and protocols a Target Service understands, Types it implements, and Scopes it is in.

2.2 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 [RFC 2119].

This specification uses the following syntax to define normative outlines for messages:

The syntax appears as an XML instance, but values in italics indicate data types instead of literal values.

Characters are appended to elements and attributes to indicate cardinality:

- "?" (0 or 1)
- "*" (0 or more)
- "+" (1 or more)

- The character "|" is used to indicate a choice between alternatives.
- The characters "[" and "]" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.

- Ellipses (i.e., "...") indicate points of extensibility. Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. If a receiver does not recognize an extension, the receiver SHOULD ignore the extension.
- XML namespace prefixes (see Table 3) are used to indicate the namespace of the element being defined.

Elsewhere in this specification, the characters "[" and "]" are used to call out references and property names. This specification uses the **[action]** and Fault properties [WS-Addressing] to define faults.

2.3 XML Namespaces

The XML Namespace URI that MUST be used by implementations of this specification is:

```
http://schemas.xmlsoap.org/ws/2005/04/discovery
```

Table 3 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Table 3: Prefixes and XML Namespaces used in this specification.

Prefix	XML Namespace	Specification(s)
s	(Either SOAP 1.1 or 1.2)	(Either SOAP 1.1 or 1.2)
s11	http://schemas.xmlsoap.org/soap/envelope/	[SOAP 1.1]
s12	http://www.w3.org/2003/05/soap-envelope	[SOAP 1.2]
a	http://schemas.xmlsoap.org/ws/2004/08/addressing	[WS-Addressing]
d	http://schemas.xmlsoap.org/ws/2005/04/discovery	This specification
ds	http://www.w3.org/2000/09/xmldsig#	[XML Sig]
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	[WS-Security]
xs	http://www.w3.org/2001/XMLSchema	[XML Schema Part 1, 2]

2.4 Protocol Assignments

If IP multicast is used to send multicast messages described herein, they MUST be sent using the following assignments:

- DISCOVERY_PORT: port 3702 [IANA]
- IPv4 multicast address: 239.255.255.250
- IPv6 multicast address: FF02::C (link-local scope)

Other address bindings may be defined but are beyond the scope of this specification.

Messages sent over UDP MUST be sent using SOAP over UDP [SOAP/UDP]. To compensate for possible UDP unreliability, senders MUST use the example transmission algorithm in Appendix I of SOAP over UDP.

As designated below, before sending some message types defined herein, a Target Service MUST wait for a timer to elapse before sending the message. This timer MUST be set to a random value between 0 and APP_MAX_DELAY. Table 4 specifies the default value for this parameter.

Table 4: Default value for an application-level transmission parameter.

Parameter	Default Value
APP_MAX_DELAY	500 milliseconds

The default value in Table 4 MAY be revised by other specifications.

Note: The authors expect this parameter to be adjusted based on interoperability test results.

Other transport bindings may be defined but are beyond the scope of this specification.

2.5 Compliance

An endpoint MAY implement more than one of the roles Target Service, Discovery Proxy, and Client; however, for each implemented, it MUST implement them as specified herein.

An implementation is not compliant with this specification if it fails to satisfy one or more of the MUST or REQUIRED level requirements defined herein for the roles it implements.

Normative text within this specification takes precedence over normative outlines, which in turn take precedence over the XML Schema [XML Schema Part 1, Part 2] and WSDL [WSDL 1.1] descriptions, which in turn take precedence over examples.

2.6 Endpoint References

As part of the discovery process, Target Services present to the network (a) a stable identifier and (b) one or more transport addresses at which network messages can be directed. This information is contained in an `a:EndpointReference` element [WS-Addressing]. Nearly all of the SOAP messages defined herein contain the `a:EndpointReference` element, a facsimile is reproduced here for convenience:

```
<a:EndpointReference>
  <a:Address>xs:anyURI</a:Address>
 [<a:ReferenceProperties> ... </a:ReferenceProperties>]?
  ...
</a:EndpointReference>
```

The combination of `a:Address` and `a:ReferenceProperties` provide a stable and globally-unique identifier.

Of particular interest is the required `a:Address` child element, which WS-Addressing specifies to contain either "a logical address or identifier", and does not require it to be a network-resolvable transport address. By convention, this specification recommends using a globally-unique identifier (GUID) as a "`uuid:`" scheme URI in this element; if the value of this element is not a network-resolvable transport address, such transport address(es) are conveyed in a separate `d:XAddrs` element defined herein (see below).

3. Model

Figure 1 depicts the message exchanges between a Target Service and a Client.

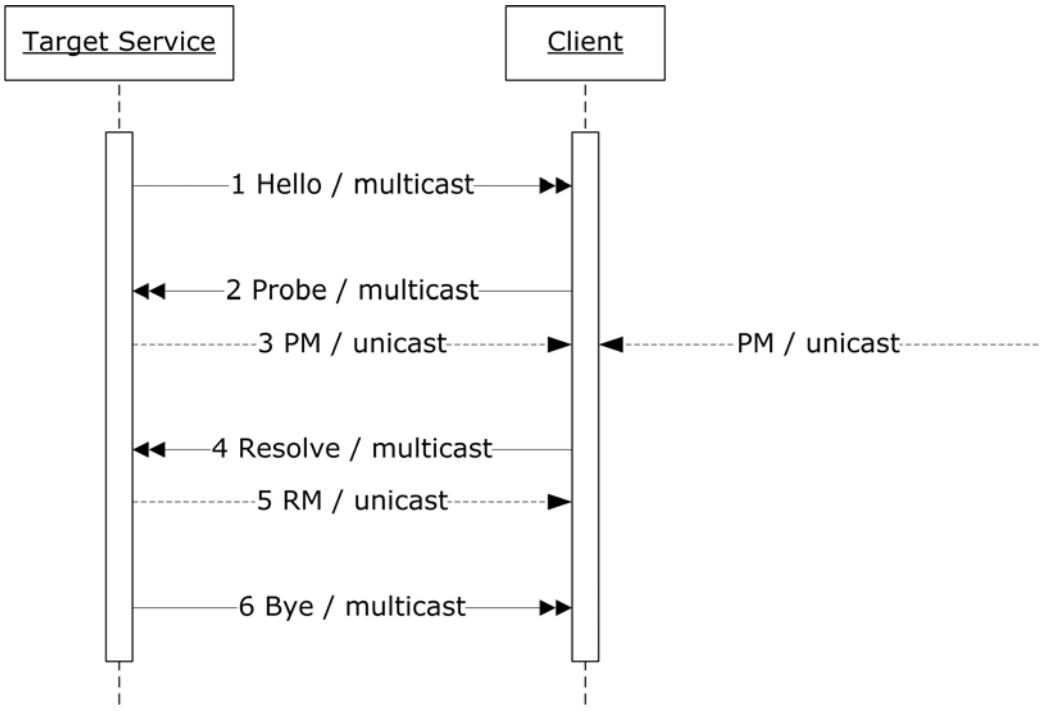

Figure 1: Message exchanges.

Starting on the left of Figure 1, initially a Target Service (1) sends a multicast Hello when it joins a network. A Target Service may (2) receive a multicast Probe at any time and (3) send a unicast Probe Match (PM) if the Target Service matches a Probe; other matching Target Services may also send unicast PM. Similarly, a Target Service may (4) receive a multicast Resolve at any time and (5) send a unicast Resolve Match (RM) if it is the target of a Resolve. Finally, when a Target Service leaves a network, it makes an effort to (6) send a multicast Bye.

Moving to the right of Figure 1, a Client mirrors Target Service messages. A Client listens to multicast Hello, may Probe to find Target Services or may Resolve to find a particular Target Service, and listens to multicast Bye.

Conceptually, Hello, Probe Match, and Resolve Match contain different kinds of information as Figure 2 depicts.

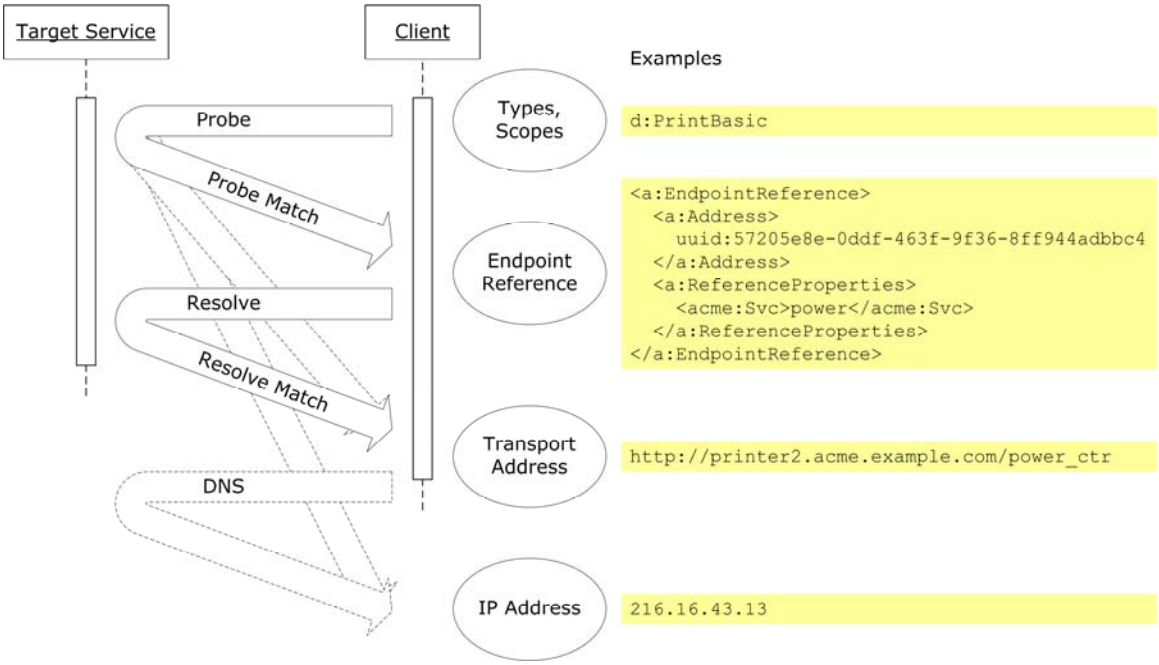

Figure 2: Conceptual content of messages.

Starting at the top of Figure 2, Probe maps from Types and/or Scopes to an Endpoint Reference [WS-Addressing]; though not depicted, Hello also provides an Endpoint Reference. Resolve maps this information to one or more transport addresses. Other address mappings may be needed, e.g., DNS, but are beyond the scope of this specification.

The required components of each message are defined in detail below, but as an optimization, a Target Service may short-circuit these message exchanges by including additional components; for instance, a Probe Match may contain transport address(es) along with an Endpoint Reference, or a transport address may use an IP address instead of a DNS name.

To limit multicast traffic, Clients operate in one of two modes as depicted in Figure 3.

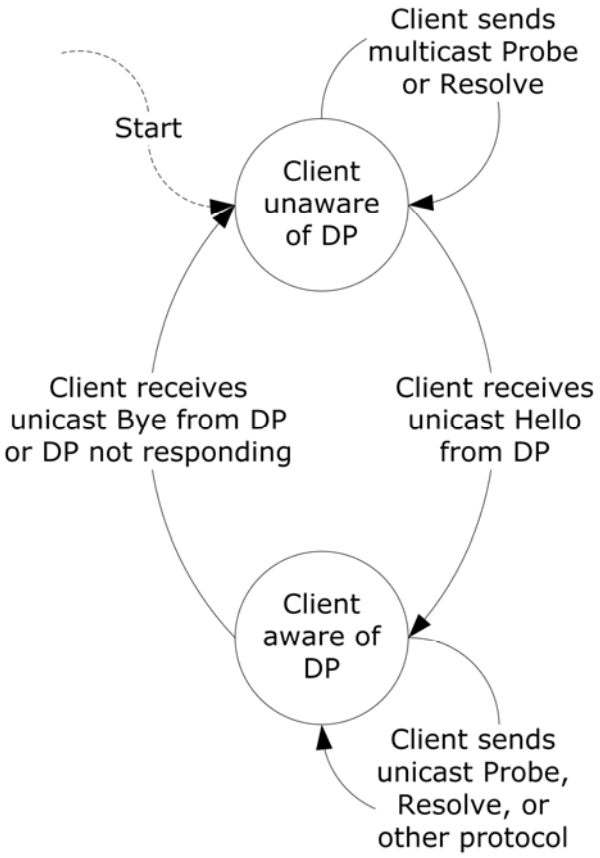

Figure 3: Client states.

By default, a new Client assumes that no Discovery Proxy (DP) is available, listens for Hello and Bye announcements, sends Probe and/or Resolve messages, and listens for Probe Match and/or Resolve Match messages as specified herein.

However, if one or more DP are available, those DP send a unicast Hello with a well-known "discovery proxy" type (described below) in response to any multicast Probe or Resolve. As depicted in Figure 4, Clients listen for this signal that one or more DP are available, and for subsequent searches, Clients do not send Probe and Resolve messages multicast but instead unicast directly to one or more DP whilst ignoring multicast Hello and Bye from Target Services.

A Client communicates with a DP using transport information contained in the DP Hello; this is typically indicated by the scheme of a transport URI, e.g., "`http:`" (HTTP), "`soap.udp:`" (UDP [SOAP/UDP]), or other.

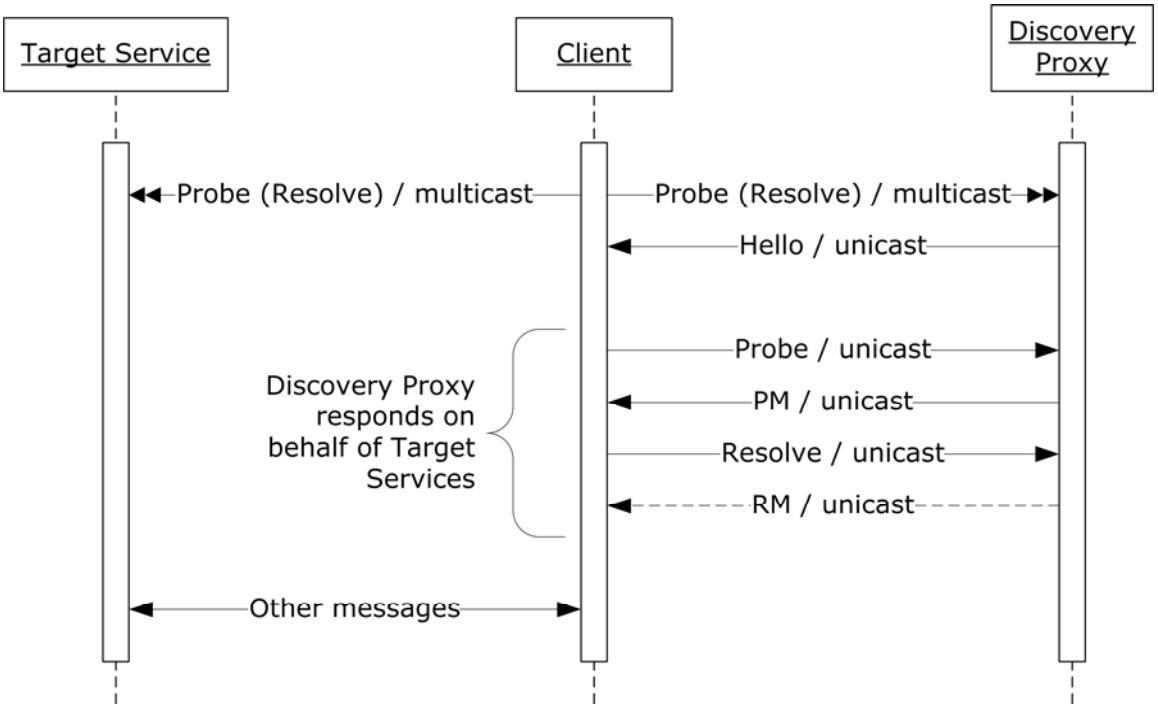

Figure 4: Discovery Proxy message exchanges.

If these DP are unresponsive after DP_MAX_TIMEOUT, or if they send a Bye, Clients revert to using the multicast messages specified herein. Table 5 specifies the default value for this parameter.

Table 5: Default value for Discovery Proxy timeout parameter.

Parameter	Default Value
DP_MAX_TIMEOUT	5 seconds

This design minimizes discovery latency in ad hoc networks without increasing multicast traffic in managed networks. To see this, note that a Client only generates multicast traffic when it sends a Probe or Resolve; while a Client could Probe (or Resolve) for a DP *before* Probing (or Resolving) for a Target Service of interest, this is just as expensive in a managed network (in terms of multicast network traffic) as allowing the Client to Probe (or Resolve) for the Target Service directly and having the DP respond to signal its presence; the reduced latency in ad hoc networks arises because the Client does not need to explicitly search and wait for possible DP responses. Some Clients (for example, mobile clients frequently moving within and beyond managed environments) may be configured to Probe first for a DP and, only if such Probe fails, switch to the operational mode described above. Specific means of such configuration is beyond of the scope of this specification.

Unlike a Client, a Target Service always sends (multicast) Hello and Bye, and always responds to Probe and Resolve with (unicast) Probe Match and Resolve Match, respectively. A Target Service does not need to explicitly recognize and/or track the availability of a DP – a Target Service behaves the same way regardless of the presence or absence of a DP. This is because the Hello and Bye are too infrequent and therefore generate too little multicast traffic to warrant adding complexity to Target Service behavior. However, some Target Services may be configured to

unicast Hello and Bye directly to a DP; these would not multicast Hello and Bye or respond to Probe or Resolve; specific means of such configuration are beyond the scope of this specification.

4. Hello and Bye

Support for messages described in this section MUST be implemented by a Target Service, MUST be implemented by a Discovery Proxy (for itself, not for other Target Services), and MAY be implemented by a Client.

4.1 Hello

A Target Service MUST send a one-way Hello when any of the following occur:

- It joins a network. This may be detected through low-level mechanisms, such as wireless beacons, or through a change in IP connectivity on one or more of its network interfaces.
- Its metadata changes (see `/s:Envelope/s:Body/*/d:MetadataVersion` below).

The Hello MUST be sent multicast using the assignments listed in Section 2.4 Protocol Assignments.

To minimize the risk of a network storm (e.g., after a network crash and recovery or power black out and restoration), a Target Service MUST wait for a timer to elapse after one of the above occurs before sending the Hello as described in Section 2.4 Protocol Assignments.

A Discovery Proxy must listen for multicast Probe (and Resolve) using the assignments listed in Section 2.4 Protocol Assignments. In response to any multicast Probe (or multicast Resolve) from a Client, a Discovery Proxy MUST send a unicast Hello to the Client and SHOULD send the Hello without waiting for a timer to elapse. The meaning of this message is that the Client MUST NOT multicast Probe (or Resolve), switch to unicast Probe (or Resolve) to the Discovery Proxy, and/or use a discovery proxy-specific protocol (see Section 3. Model).

The normative outline for Hello is:

```
<s:Envelope ... >
  <s:Header ... >
    <a:Action ... >
      http://schemas.xmlsoap.org/ws/2005/04/discovery/Hello
    </a:Action>
    <a:MessageID ... >xs:anyURI</a:MessageID>
   [<a:RelatesTo RelationshipType="d:Suppression" >
      xs:anyURI
    </a:RelatesTo>]?
    <a:To ... >urn:schemas-xmlsoap-org:ws:2005:04:discovery</a:To>
    <d:AppSequence ... />
    ...
  </s:Header>
```

```
  <s:Body ... >
    <d:Hello ... >
      <a:EndpointReference> ... </a:EndpointReference>
     [<d:Types>list of xs:QName</d:Types>]?
     [<d:Scopes>list of xs:anyURI</d:Scopes>]?
     [<d:XAddrs>list of xs:anyURI</d:XAddrs>]?
      <d:MetadataVersion>xs:unsignedInt</d:MetadataVersion>
      ...
    </d:Hello>
  </s:Body>
</s:Envelope>
```

The following describes additional normative constraints on the outline listed above:

/s:Envelope/s:Header/*
Per SOAP [SOAP 1.1, SOAP 1.2], header blocks MAY appear in any order.

/s:Envelope/s:Header/a:RelatesTo
MUST be included only by a Discovery Proxy and if and only if Hello is sent unicast in response to a multicast Probe (or Resolve). It MUST be the value of the **[message id]** property [WS-Addressing] of the multicast Probe (Resolve).

/s:Envelope/s:Header/a:RelatesTo/@RelationshipType="d:Suppression"
Indicates this message is a suppression of the multicast Probe (or Resolve).

/s:Envelope/s:Header/d:AppSequence
MUST be included to allow ordering discovery messages from a Target Service (see Appendix I – Application Sequencing).

/s:Envelope/s:Body/*/a:EndpointReference
Endpoint Reference for the Target Service (see Section 2.6 Endpoint References).

/s:Envelope/s:Body/*/d:Types
Unordered set of Types implemented by the Target Service (or Discovery Proxy).

- For a Target Service, if omitted, no implied value.
- For a Discovery Proxy, MUST be included and MUST explicitly include `d:DiscoveryProxy` and `d:TargetService`. The former indicates it is a Discovery Proxy, and the latter indicates it supports Target Service messages at this Endpoint Reference and transport address(es) (see ./d:XAddrs).

/s:Envelope/s:Body/*/d:Scopes
Unordered set of Scopes the Target Service (or Discovery Proxy) is in, which MAY be of more than one URI scheme. If included, MUST be a set of absolute URIs, and contained URIs MUST NOT contain white space. If omitted, implied value is a set that includes "`http://schemas.xmlsoap.org/ws/2005/04/discovery/adhoc`".

/s:Envelope/s:Body/*/d:XAddrs
Transport address(es) that MAY be used to communicate with the Target Service (or Discovery Proxy). Contained URIs MUST NOT contain white space.

/s:Envelope/s:Body/*/d:MetadataVersion

Incremented by >= 1 whenever there is a change in the metadata of the Target Service. If a Target Service goes down and comes back up again, this value MAY be incremented but MUST NOT be decremented (see Appendix I – Application Sequencing). Metadata includes, but is not limited to, ../d:Types and ../d:Scopes. By design, this value MAY be used by the Client and/or Discovery Proxy for cache control of Target Service metadata.

To minimize the need to Probe, Clients SHOULD listen for Hello messages and store (or update) information for the corresponding Target Service. Note that a Target Service MAY vary the amount of metadata it includes in Hello messages (or Probe Match or Resolve Match messages), and consequently, a Client may receive two such messages containing the same /s:Envelope/s:Body/*/d:MetadataVersion but containing different metadata. If a Client chooses to cache metadata, it MAY, but is not constrained to, adopt any of the following behaviors:

- Cache the union of the previously cached and new metadata.
- Replace the previously cached with new metadata.
- Use some other means to retrieve more complete metadata.

However, to prevent network storms, a Client SHOULD NOT delete cached metadata and SHOULD NOT repeat a Probe (or Resolve) if it detects differences in contained metadata.

Table 6 lists an example Hello for the same Target Service that responded with a Probe Match in Table 2.

Table 6: Example Hello.

```
(01) <s:Envelope
(02)   xmlns:a="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(03)   xmlns:d="http://schemas.xmlsoap.org/ws/2005/04/discovery"
(04)   xmlns:s="http://www.w3.org/2003/05/soap-envelope" >
(05)   <s:Header>
(06)     <a:Action>
(07)       http://schemas.xmlsoap.org/ws/2005/04/discovery/Hello
(08)     </a:Action>
(09)     <a:MessageID>
(10)       uuid:73948edc-3204-4455-bae2-7c7d0ff6c37c
(11)     </a:MessageID>
(12)     <a:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</a:To>
(13)     <d:AppSequence InstanceId="1077004800" MessageNumber="1" />
(14)   </s:Header>
(15)   <s:Body>
(16)     <d:Hello>
(17)       <a:EndpointReference>
(18)         <a:Address>
(19)           uuid:98190dc2-0890-4ef8-ac9a-5940995e6119
(20)         </a:Address>
(21)       </a:EndpointReference>
(22)       <d:MetadataVersion>75965</d:MetadataVersion>
```

```
(23)     </d:Hello>
(24)   </s:Body>
(25) </s:Envelope>
(26)
```

Lines (06-08) indicate this is a Hello, and because Line (12) is set to the distinguished URI defined herein, this is a multicast Hello. Line (13) contains an instance identifier as well as a message number; this information allows the receiver to reorder Hello and Bye messages from a Target Service. Lines (17-21) are identical to the corresponding lines in the Probe Match in Table 2.

4.2 Bye

A Target Service SHOULD send a one-way Bye message when it is preparing to leave a network. (A Target Service MUST NOT send a Bye message when its metadata changes.)

The Bye MUST be sent multicast using the assignments listed in Section 2.4 Protocol Assignments.

A Target Service MAY send the Bye without waiting for a timer to elapse.

The normative outline for Bye is:

```
<s:Envelope ... >
  <s:Header ... >
    <a:Action ... >
      http://schemas.xmlsoap.org/ws/2005/04/discovery/Bye
    </a:Action>
    <a:MessageID ... >xs:anyURI</a:MessageID>
    <a:To ...>urn:schemas-xmlsoap-org:ws:2005:04:discovery</a:To>
    <d:AppSequence ... />
    ...
  </s:Header>
  <s:Body ... >
    <d:Bye ... >
      <a:EndpointReference> ... </a:EndpointReference>
      ...
    </d:Bye>
  </s:Body>
</s:Envelope>
```

The following describes additional normative constraints on the outline listed above:

/s:Envelope/s:Header/*
Per SOAP [SOAP 1.1, SOAP 1.2], header blocks MAY appear in any order.

/s:Envelope/s:Body/*/a:EndpointReference
Endpoint Reference for the Target Service (see Section 2.6 Endpoint References).

Clients SHOULD listen for Bye messages, marking or removing corresponding information as invalid. Clients MAY wish to retain information associated with a Target Service that has left the network, for instance if the Client expects the Target Service to rejoin the network at some point in the future. Conversely, Clients MAY discard information associated with a Target Service at any time, based on, for instance, preset maximums on the amount of memory allocated for this use, lack of communication to the Target Service, preferences for other Target Service Types or Scopes, and/or other application-specific preferences.

Table 7 lists an example Bye message corresponding to the Hello in Table 6.

Table 7: Example Bye.

```
(01) <s:Envelope
(02)     xmlns:a="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(03)     xmlns:d="http://schemas.xmlsoap.org/ws/2005/04/discovery"
(04)     xmlns:s="http://www.w3.org/2003/05/soap-envelope" >
(05)   <s:Header>
(06)     <a:Action>
(07)       http://schemas.xmlsoap.org/ws/2005/04/discovery/Bye
(08)     </a:Action>
(09)     <a:MessageID>
(10)       uuid:337497fa-3b10-43a5-95c2-186461d72c9e
(11)     </a:MessageID>
(12)     <a:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</a:To>
(13)     <d:AppSequence InstanceId="1077004800" MessageNumber="4" />
(14)   </s:Header>
(15)   <s:Body>
(16)     <d:Bye>
(17)       <a:EndpointReference>
(18)         <a:Address>
(19)           uuid:98190dc2-0890-4ef8-ac9a-5940995e6119
(20)         </a:Address>
(21)       </a:EndpointReference>
(22)     </d:Bye>
(23)   </s:Body>
(24) </s:Envelope>
(25)
```

Lines (06-08) indicate this is a Bye, and like the Hello in Table 6, the distinguished URI in Line (12) indicates it is a multicast Bye sent over the multicast channels listed in Section 2.4 Protocol Assignments. The sequence information in Line (13) indicates this message is to be ordered after the Hello in Table 6 because the Bye has a larger message number than the Hello within the same instance identifier. Note that the Body (Lines 16-22) is an abbreviated form of the corresponding information in the Hello; when a Target Service leaves a network, it is sufficient to send the stable identifier to indicate the Target Service is no longer available.

5. Probe and Probe Match

To find Target Services by the Type of the Target Service, a Scope in which the Target Service resides, both, or simply all Target Services, a Client sends a Probe.

Support for messages described in this section MUST be implemented by a Target Service, MUST be implemented by a Discovery Proxy (for itself and for other Target Services), and MAY be implemented by a Client.

5.1 Matching Types and Scopes

A Probe includes zero, one, or two constraints on matching Target Services: a set of Types and/or a set of Scopes. A Probe Match MUST include a Target Service if and only if all of the Types and all of the Scopes in the Probe match the Target Service.

A Type T1 in a Probe matches Type T2 of a Target Service if the QNames match. Specifically, T1 matches T2 if all of the following are true:

- The namespace [Namespaces in XML 1.1] of T1 and T2 are the same.
- The local name of T1 and T2 are the same.

(The namespace prefix of T1 and T2 is relevant only to the extent that it identifies the namespace.)

A Scope S1 in a Probe matches Scope S2 of a Target Service per the rule indicated within the Probe. This specification defines the following matching rules. Other matching rules MAY be used, but if a matching rule is not recognized by a receiver of the Probe, S1 does not match S2 regardless of the value of S1 and/or S2.

http://schemas.xmlsoap.org/ws/2005/04/discovery/rfc2396

Using a case-insensitive comparison,

- The `scheme` [RFC 2396] of S1 and S2 is the same and
- The `authority` of S1 and S2 is the same and

Using a case-sensitive comparison,

- The `path_segments` of S1 is a `segment`-wise (not string) prefix of the `path_segments` of S2 and
- Neither S1 nor S2 contain the `"."` `segment` or the `".."` `segment`.

All other components (e.g., `query` and `fragment`) are explicitly excluded from comparison. S1 and S2 MUST be canonicalized (e.g., unescaping escaped characters) before using this matching rule.

Note: this matching rule does NOT test whether the string representation of S1 is a prefix of the string representation of S2. For example, "http://example.com/abc" matches "http://example.com/abc/def" using this rule but "http://example.com/a" does not.

http://schemas.xmlsoap.org/ws/2005/04/discovery/uuid

Using a case-insensitive comparison, the scheme of S1 and S2 is `"uuid"` and each of the unsigned integer fields [UUID] in S1 is equal to the corresponding field in S2, or equivalently, the 128 bits of the in-memory representation of S1 and S2 are the same 128 bit unsigned integer.

http://schemas.xmlsoap.org/ws/2005/04/discovery/ldap

Using a case-insensitive comparison, the scheme of S1 and S2 is `"ldap"` and the `hostport` [RFC 2255] of S1 and S2 is the same and the `RDNSequence` [RFC 2253] of the dn of S1 is a prefix of the `RDNSequence` of the dn of S2, where comparison

does not support the variants in an RDNSequence described in Section 4 of RFC 2253 [RFC 2253].

http://schemas.xmlsoap.org/ws/2005/04/discovery/strcmp0
: Using a case-sensitive comparison, the string representation of S1 and S2 is the same.

5.2 Probe

A Client MAY send a Probe to find Target Services of a given Type and/or in a given Scope or to find Target Services regardless of their Types or Scopes.

A Probe is a one-way message.

If a Client has not detected any Discovery Proxies, the Probe is sent multicast using the assignments listed in Section 2.4 Protocol Assignments.

If a Client knows a transport address of a Target Service, the Probe MAY be sent unicast to that address.

Because a Client may not know in advance how many Target Services (if any) will send Probe Match, the Client MAY adopt either of the following behaviors:

- Wait for a sufficient number of Probe Match messages.
- Repeat the Probe several times until the Client is convinced that no further Probe Match messages will be received. The Client MUST use the same value for the **[message id]** property [WS-Addressing] in all copies of the Probe.

If a Client has detected a Discovery Proxy, the Probe is sent unicast to the Discovery Proxy.

The normative outline for Probe is:

```
<s:Envelope ... >
  <s:Header ... >
    <a:Action ... >
      http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe
    </a:Action>
    <a:MessageID ... >xs:anyURI</a:MessageID>
   [<a:ReplyTo ... >endpoint-reference</a:ReplyTo>]?
    <a:To ... >xs:anyURI</a:To>
    ...
  </s:Header>
  <s:Body ... >
    <d:Probe ... >
     [<d:Types>list of xs:QName</d:Types>]?
     [<d:Scopes [MatchBy="xs:anyURI"]? ... >
        list of xs:anyURI
      </d:Scopes>]?
     ...
```

```
    </d:Probe>
  </s:Body>
</s:Envelope>
```

The following describes additional normative constraints on the outline listed above:

/s:Envelope/s:Header/*

Per SOAP [SOAP 1.1, SOAP 1.2], header blocks MAY appear in any order.

/s:Envelope/s:Header/a:ReplyTo

If included, MUST be of type a:EndpointReferenceType [WS-Addressing]. If omitted, implied value of the **[reply endpoint]** property [WS-Addressing] is "`http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous`".

/s:Envelope/s:Header/a:ReplyTo/a:Address

If the value is "`http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous`", **[reply endpoint]** property is defined by the underlying transport. If the Probe was received over UDP, the **[reply endpoint]** is the IP source address and port number of the Probe transport header [SOAP/UDP].

/s:Envelope/s:Header/a:To

- If sent to a Target Service, MUST be "`urn:schemas-xmlsoap-org:ws:2005:04:discovery`" [RFC 2141].
- If sent to a Discovery Proxy, MUST be the **[address]** property of the Endpoint Reference for the Discovery Proxy, e.g., as contained in a Hello from the Discovery Proxy.

/s:Envelope/s:Body/d:Probe/d:Types

If omitted, implied value is any Type.

/s:Envelope/s:Body/d:Probe/d:Scopes

If included, MUST be a list of absolute URIs. If omitted, implied value is any Scope.

/s:Envelope/s:Body/d:Probe/d:Scopes/@MatchBy

If omitted, implied value is "`http://schemas.xmlsoap.org/ws/2005/04/discovery/rfc2396`".

If a Target Service or Discovery Proxy receives a unicast Probe and does not support the matching rule, it MAY choose not to send a Probe Match and instead generate a fault, bound to SOAP [WS-Addressing] as follows:

[action]	http://schemas.xmlsoap.org/ws/2005/04/discovery/fault
[Code]	s12:Sender
[Subcode]	d:MatchingRuleNotSupported
[Reason]	E.g., the matching rule specified is not supported.

[Detail]	`<d:SupportedMatchingRules>`
	`list of xs:anyURI`
	`</d:SupportedMatchingRules>`

To Probe for all Target Services, a Client MAY omit both `/s:Envelope/s:Body/d:Probe/d:Types` and `./d:Scopes`.

5.3 Probe Match

If a Target Service matches a Probe, the Target Service MUST respond with a Probe Match message. If the Target Service receives more than one copy of the Probe, it SHOULD respond only once. (The transport may require transport-level retransmission, e.g., *_UDP_REPEAT [SOAP/UDP].) A Target Service MUST wait for a timer to elapse after receiving a Probe before sending a Probe Match as described in Section 2.4 Protocol Assignments.

If a Target Service receives a Probe and does not match the Probe, it MUST NOT respond with a Probe Match.

If a Discovery Proxy receives a Probe by multicast, it MUST respond with a Hello (see Section 4.1 Hello).

A Discovery Proxy MUST respond with a Probe Match message without waiting for a timer to elapse. However, the Probe Match MAY contain zero matches if the Discovery Proxy has no matching Target Services.

A Probe Match MUST be unicast to the **[reply endpoint]** property [WS-Addressing] of the Probe.

The normative outline for Probe Match is:

```
<s:Envelope ... >
  <s:Header ... >
    <a:Action ... >
      http://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches
    </a:Action>
    <a:MessageID ... >xs:anyURI</a:MessageID>
    <a:RelatesTo ... >xs:anyURI</a:RelatesTo>
    <a:To ... >xs:anyURI</a:To>
    <d:AppSequence ... />
    ...
  </s:Header>
  <s:Body ... >
    <d:ProbeMatches ... >
     [<d:ProbeMatch ... >
        <a:EndpointReference> ... </a:EndpointReference>
       [<d:Types>list of xs:QName</d:Types>]?
       [<d:Scopes>list of xs:anyURI</d:Scopes>]?
       [<d:XAddrs>list of xs:anyURI</d:XAddrs>]?
        <d:MetadataVersion>xs:unsignedInt</d:MetadataVersion>
        ...
      </d:ProbeMatch>]*
      ...
```

```
    </d:ProbeMatches>
  </s:Body>
</s:Envelope>
```

The following describes additional normative constraints on the outline listed above:

/s:Envelope/s:Header/*
Per SOAP [SOAP 1.1, SOAP 1.2], header blocks MAY appear in any order.

/s:Envelope/s:Header/a:RelatesTo
MUST be the value of the **[message id]** property [WS-Addressing] of the Probe.

/s:Envelope/s:Header/a:To
If the **[reply endpoint]** property [WS-Addressing] of the corresponding Probe is the IP source address and port number of the Probe transport header (e.g., when the a:ReplyTo header block was omitted from the corresponding Probe), the value of this header block MUST be "`http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous`".

/s:Envelope/s:Header/d:AppSequence
MUST be included to allow ordering discovery messages from a Target Service (see Appendix I – Application Sequencing).

/s:Envelope/s:Body/d:ProbeMatches
Matching Target Services.

- If this Probe Match was sent by a Target Service, this element will contain one `d:ProbeMatch` child. (If Target Service doesn't match the Probe, the Target Service does not send a Probe Match at all.)
- If this Probe Match was sent by a Discovery Proxy, this element will contain zero or more `d:ProbeMatch` children. (Discovery Proxies always respond to Probe.)

/s:Envelope/s:Body/d:ProbeMatches/d:ProbeMatch/a:EndpointReference
Endpoint Reference for the Target Service (see Section 2.6 Endpoint References).

/s:Envelope/s:Body/d:ProbeMatches/d:ProbeMatch/d:Types
See /s:Envelope/s:Body/*/d:Types in Section 4.1 Hello.

/s:Envelope/s:Body/d:ProbeMatches/d:ProbeMatch/d:Scopes
See /s:Envelope/s:Body/*/d:Scopes in Section 4.1 Hello.

/s:Envelope/s:Body/d:ProbeMatches/d:ProbeMatch/d:XAddrs
See /s:Envelope/s:Body/*/d:XAddrs in Section 4.1 Hello.

/s:Envelope/s:Body/d:ProbeMatches/d:ProbeMatch/d:MetadataVersion
See /s:Envelope/s:Body/*/d:MetadataVersion in Section 4.1 Hello.

6. Resolve and Resolve Match

To locate a Target Service, i.e., to retrieve its transport address(es), a Client sends a Resolve.

Support for messages described in this section MUST be implemented by a Target Service, MUST be implemented by a Discovery Proxy (for itself and for other Target Services), and MAY be implemented by a Client.

6.1 Resolve

A Client MAY send a Resolve to retrieve network transport information for a Target Service if it has an Endpoint Reference [WS-Addressing] for the Target Service.

A Resolve is a one-way message.

If a Client has not detected any Discovery Proxies, the Resolve is sent multicast using the assignments listed in Section 2.4 Protocol Assignments.

If a Client has detected a Discovery Proxy, the Resolve is sent unicast to the Discovery Proxy.

The normative outline for Resolve is:

```
<s:Envelope ... >
  <s:Header ... >
    <a:Action ... >
      http://schemas.xmlsoap.org/ws/2005/04/discovery/Resolve
    </a:Action>
    <a:MessageID ... >xs:anyURI</a:MessageID>
   [<a:ReplyTo ... >endpoint-reference</a:ReplyTo>]?
    <a:To ... >xs:anyURI</a:To>
    ...
  </s:Header>
  <s:Body>
    <d:Resolve ... >
      <a:EndpointReference> ... </a:EndpointReference>
      ...
    </d:Resolve>
  </s:Body>
</s:Envelope>
```

The following describes additional normative constraints on the outline above:

/s:Envelope/s:Header/*
Per SOAP [SOAP 1.1, SOAP 1.2], header blocks MAY appear in any order.

/s:Envelope/s:Header/a:ReplyTo
As constrained for Probe (see Section 5.2 Probe).

/s:Envelope/s:Header/a:To
As constrained for Probe (see Section 5.2 Probe).

/s:Envelope/s:Body/*/a:EndpointReference
Endpoint Reference for the Target Service (see Section 2.6 Endpoint References).

6.2 Resolve Match

If a Target Service matches a Resolve, the Target Service MUST respond with a Resolve Match message. Comparison MUST be done per WS-Addressing Section 2.4 Endpoint Reference Comparison [WS-Addressing]. If the Target Service receives

more than one copy of the Resolve, it SHOULD respond only once. (The transport may require transport-level retransmission, e.g., *_UDP_REPEAT [SOAP/UDP].)

If a Target Service receives a Resolve and does not match the Resolve, it MUST NOT respond with a Resolve Match.

If a Discovery Proxy receives a Probe by multicast, it MUST respond with a Hello (see Section 4.1 Hello).

If a Discovery Proxy has a Target Service that matches a Resolve, the Discovery Proxy MUST respond with a Resolve Match message. However, the Resolve Match MAY contain zero matches if the Discovery Proxy has no matching Target Service.

A Resolve Match MUST be unicast to the **[reply endpoint]** property [WS-Addressing] of the Resolve without waiting for a timer to elapse.

The normative outline for Resolve Match is:

```
<s:Envelope ... >
  <s:Header ... >
    <a:Action ... >
      http://schemas.xmlsoap.org/ws/2005/04/discovery/ResolveMatches
    </a:Action>
    <a:MessageID ... >xs:anyURI</a:MessageID>
    <a:RelatesTo ... >xs:anyURI</a:RelatesTo>
    <a:To ... >xs:anyURI</a:To>
    <d:AppSequence ... />
    ...
  </s:Header>
  <s:Body ... >
    <d:ResolveMatches ... >
     [<d:ResolveMatch ... >
        <a:EndpointReference> ... </a:EndpointReference>
       [<d:Types>list of xs:QName</d:Types>]?
       [<d:Scopes>list of xs:anyURI</d:Scopes>]?
        <d:XAddrslist of xs:anyURI</d:XAddrs>
        <d:MetadataVersion>xs:unsignedInt</d:MetadataVersion>
        ...
      </d:ResolveMatch>]?
      ...
    </d:ResolveMatches>
  </s:Body>
</s:Envelope>
```

The following describes additional normative constraints on the outline listed above:

/s:Envelope/s:Header/*
Per SOAP [SOAP 1.1, SOAP 1.2], header blocks MAY appear in any order.

/s:Envelope/s:Header/a:RelatesTo
MUST be the value of the **[message id]** property [WS-Addressing] of the Resolve.

/s:Envelope/s:Header/a:To
As constrained for Probe Match (see Section 5.3 Probe Match).

/s:Envelope/s:Header/d:AppSequence
As constrained for Probe Match (see Section 5.3 Probe Match).

/s:Envelope/s:Body/d:ResolveMatches
Matching Target Service.

/s:Envelope/s:Body/d:ResolveMatches/d:ResolveMatch/a:EndpointReference
Endpoint Reference for the Target Service (see Section 2.6 Endpoint References).

/s:Envelope/s:Body/d:ResolveMatches/d:ResolveMatch/d:Types
See /s:Envelope/s:Body/*/d:Types in Section 4.1 Hello.

/s:Envelope/s:Body/d:ResolveMatches/d:ResolveMatch/d:Scopes
See /s:Envelope/s:Body/*/d:Types in Section 4.1 Hello.

/s:Envelope/s:Body/d:ResolveMatches/d:ResolveMatch/d:XAddrs
See /s:Envelope/s:Body/*/d:Types in Section 4.1 Hello.

/s:Envelope/s:Body/d:ResolveMatches/d:ResolveMatch/d:MetadataVersion
See /s:Envelope/s:Body/*/d:Types in Section 4.1 Hello.

7. Security Model

This specification does not require that endpoints participating in the discovery process be secure. However, this specification RECOMMENDS that security be used to mitigate various types of attacks (see Section 9. Security Considerations).

If a Target Service wishes to secure Hello, Bye, Probe Match and/or Resolve Match, it SHOULD use the compact signature format defined in Section 8. Compact Signature Format. A Client MAY choose to ignore Hello, Bye, Probe Match, and/or Resolve Match if it cannot verify the signature.

If a Client wishes to secure Probe and Resolve, it SHOULD use the compact signature format defined in Section 8. Compact Signature Format. A Target Service MAY chose to ignore received Probe and/or Resolve if it cannot verify the signature.

There is no requirement for a Target Service to respond to a Probe (or Resolve) if any of the following are true:

- The Target Service is in a different administrative domain than the Client, and the Probe (or Resolve) was sent as multicast, or
- The Target Service fails to verify the signature contained in the Probe (or Resolve).

To avoid participating in a Distributed Denial of Service attack, a Target Service or Discovery Proxy SHOULD NOT respond to a message without a valid signature and MUST NOT respond to a message without a valid signature if the **[reply endpoint]** is not "http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous".

A Client MAY discard a Probe Match (or Resolve Match) if any of the following are true:

- The Probe Match (or Resolve Match) is received MATCH_TIMEOUT seconds or more later than the last corresponding Probe was sent, or
- The Client fails to verify the signature contained in the Probe Match (or Resolve Match).

Table 8 specifies the default value for the MATCH_TIMEOUT parameter.

Table 8: Default value for an application-level parameter.

Parameter	Default Value
MATCH_TIMEOUT	APP_MAX_DELAY + 100 milliseconds

If a Target Service has multiple credentials, it SHOULD send separate Hello, Bye, Probe Match, and/or Resolve Match using different credentials to sign each.

The same security requirements as defined for a Target Service apply to a Discovery Proxy.

8. Compact Signature Format

This section defines the signature format for signing UDP unicast and multicast messages.

To minimize the number of XML namespace declarations in messages, the following global attribute is defined:

@d:Id
: An alternate ID reference mechanism with the same meaning as @wsu:Id [WS-Security].

This attribute MAY be used to identify which message parts are signed by the compact signature.

The compact signature itself is of the following form:

```
<d:Security ... >
  [<d:Sig Scheme="xs:anyURI"
         [KeyId="xs:base64Binary"]?
          Refs="..."
          Sig="xs:base64Binary"
          ... />]?
  ...
</d:Security>
```

d:Security
: A sub-class of the wsse:Security header block [WS-Security] that has the same processing model and rules but is restricted in terms of content and usage. The d:Sig child element provides a compact message signature. Its format is a compact form of XML Signature. To process the signature, the compact form is parsed, and an XML Signature ds:SignedInfo block is created and used for signature verification.

d:Security/@s11:mustUnderstand | d:Security/@s12:mustUnderstand

Processing of the d:Security header block is not mandatory; therefore, the d:Security header block SHOULD NOT be marked mustUnderstand with a value of "true".

d:Security/d:Sig/@Scheme
: The governing scheme of the signature. Provides exactly one algorithm for digests and signatures.

d:Security/d:Sig/@Scheme = "http://schemas.xmlsoap.org/ws/2005/04/discovery/rsa"
: Exclusive C14N is used for all canonicalization, SHA1 is used for all digests, and Signatures use RSA. Specifically:

- http://www.w3.org/2001/10/xml-exc-c14n#
- http://www.w3.org/2000/09/xmldsig#sha1
- http://www.w3.org/2000/09/xmldsig#rsa-sha1

d:Security/d:Sig/@KeyId
: The key identifier of the signing token. MUST be specified if a public key token is used. If omitted, the semantics are undefined.

d:Security/d:Sig/@Refs
: Parts of the message that have been canonicalized and digested. Each part is referenced by @d:Id (see above). Only immediate children of the security header, top-level SOAP header blocks (/s:Envelope/s:Header/*), and the full SOAP Body (/s:Envelope/s:Body) can be referenced in this list. The value is a space-separated list of IDs to elements within the message.

d:Security/d:Sig/@Sig
: The value of the signature.

Table 9 lists an example compact signature.

Table 9: Example compact signature.

```
(01) <d:Sig xmlns:d="http://schemas.xmlsoap.org/ws/2005/04/discovery"
(02)        Scheme="http://schemas.xmlsoap.org/ws/2005/04/discovery/rsa"
(03)        KeyId="Dx42/9g="
(04)        Refs="ID1"
(05)        Sig="ru5Ef76xGz5Y5IB2iAzDuMvR5Tg=" />
(06)
```

A compact signature is expanded into an XML Signature ds:SignedInfo using the following pseudo-code.

1. Create an XML Signature ds:SignedInfo block. Because canonicalization includes the namespace prefix, this MUST use an XML namespace prefix of "ds" so each party can compute a consistent digest value.
2. Populate the block with the appropriate canonicalization and algorithm blocks based on the scheme in d:Security/d:Sig/@Scheme.
   - First add a ds:CanonicalizationMethod element.
   - Next add a ds:SignatureMethod element.
3. For each ID in d:Security/d:Sig/@Refs create a corresponding XML Signature Reference element to the identified part (using URI fragments) annotated with the canonicalization and digest algorithms from the scheme in

d:Security/d:Sig/@Scheme. Note that individual digests need to be computed on the fly.

- Add a ds:Reference element.
- The @URI attribute's value is a "#" followed by the specified ID.
- Inside the ds:Reference element add a ds:Transforms element that contains a ds:Transform element indicating the selected canonicalization algorithm.
- Inside the ds:Reference element add a ds:DigestMethod element.
- Inside the ds:Reference element add a ds:DigestValue element.

4. Compute the final signature, and verify that it matches.
5. d:Security/d:Sig/@KeyId, if present, can be processed as a SecurityTokenReference [WS-Security] with an embedded KeyIdentifier [WS-Security] specifying the indicated value. While it isn't required to construct a wsse:SecurityTokenReference element, the following steps illustrate how one would be created:
   - Create a wsse:SecurityTokenReference element.
   - Within this, add a wsse:KeyIdentifier element with the value of the KeyId attribute's value.

Table 10 lists the expanded form corresponding to the compact form in Table 9.

Table 10: Example expanded signature.

```
(01) <ds:Signature
(02)     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
(03)     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
     200401-wss-wssecurity-secext-1.0.xsd" >
(04)   <ds:SignedInfo>
(05)     <ds:CanonicalizationMethod
(06)         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
(07)     <ds:SignatureMethod
(08)         Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
(09)     <ds:Reference URI="#ID1" >
(10)       <ds:Transforms>
(11)         <ds:Transform
(12)             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
(13)       </ds:Transforms>
(14)       <ds:DigestMethod
(15)           Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
(16)       <ds:DigestValue>ODE3NDkyNzI5</ds:DigestValue>
(17)     </ds:Reference>
(18)   </ds:SignedInfo>
(19)   <ds:SignatureValue>
(20)       ru5Ef76xGz5Y5IB2iAzDuMvR5Tg=
(21)   </ds:SignatureValue>
(22)   <ds:KeyInfo>
```

```
(23)     <wsse:SecurityTokenReference>
(24)       <wsse:KeyIdentifier>Dx42/9g=</wsse:KeyIdentifier>
(25)     </wsse:SecurityTokenReference>
(26)   </ds:KeyInfo>
(27) </ds:Signature>
(28)
```

9. Security Considerations

Message discovery, both announcements and searches, are subject to a wide variety of attacks. Therefore communication should be secured using the mechanisms described in Section 8. Compact Signature Format.

The following list summarizes common classes of attacks and mitigations provided by this protocol:

- **Message alteration** – Message content may be changed by an attacker. To prevent this, the message should be signed. The Body and all relevant headers should be included in the signature. Specifically, the WS-Addressing [WS-Addressing] headers and any headers identified in Endpoint References should be signed together with the Body to "bind" them together.
- **Availability (Denial of Service)** – An attacker may send messages that consume resources. To prevent this, a signature assures that a message is of genuine origin. To avoid unnecessary processing, the signature should be validated before performing beginning any significant processing of message content.
- **Replay** – An attacker may resend a valid message and cause duplicate processing. To prevent this, a replayed message is detected by a duplicate **[message id]** property [WS-Addressing] and should be discarded.
- **Spoofing** – An attacker sends a message that pretends to be of genuine origin. To prevent this, the signature should be unique to the sender.

To provide mitigation against other possible attacks, e.g., message disclosure, mechanisms defined in WS-Security [WS-Security], WS-SecureConversation [WS-SecureConversation], and/or WS-Trust [WS-Trust] may be applied.

If a Client communicates with a Discovery Proxy, the Client should establish end-to-end security with the Discovery Proxy; to improve the efficiency of security operations, the Client should establish a security context using the mechanisms described in WS-Trust [WS-Trust] and WS-SecureConversation [WS-SecureConversation]. In such cases, separate derived keys should be used to secure each message.

10. Acknowledgements

This specification has been developed as a result of joint work with many individuals and teams, including: Don Box (Microsoft), Shannon Chan (Microsoft), Dan Conti (Microsoft), Ken Cooper (Microsoft), Mike Fenelon (Microsoft), Omri Gazitt (Microsoft), Bertus Greeff (Microsoft), Rob Hain (Microsoft), Richard Hasha (Microsoft), Erin Honeycutt (Microsoft), Christian Huitema (Microsoft), Chris Kaler (Microsoft), Umesh Madan (Microsoft), Vipul Modi (Microsoft), Jeff Parham (Microsoft), Yaniv Pessach (Microsoft), Stefan Pharies (Microsoft), Dale Sather (Microsoft), and Matt Tavis (Microsoft).

11. References

[IANA]

"Port Numbers," February 2005. (See http://www.iana.org/assignments/port-numbers.)

[Namespaces in XML 1.1]

T. Bray, et al, "Namespaces in XML 1.1," February 2004. (See http://www.w3.org/TR/2004/REC-xml-names11-20040204/.)

[RFC 2119]

S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," March 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

[RFC 2141]

R. Moats, "URN Syntax," May 1997. (See http://www.ietf.org/rfc/rfc2141.txt.)

[RFC 2253]

M. Wahl, et al, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names," December 1997. (See http://www.ietf.org/rfc/rfc2253.txt.)

[RFC 2255]

T. Howes, et al, "The LDAP URL Format," December 1997. (See http://www.ietf.org/rfc/rfc2255.txt.)

[RFC 2396]

T. Berners-Lee, et al, "Uniform Resource Identifiers (URI): Generic Syntax," August 1998. (See http://www.ietf.org/rfc/rfc2396.txt.)

[SOAP 1.1]

D. Box, et al, "Simple Object Access Protocol (SOAP) 1.1," May 2000. (See http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.)

[SOAP 1.2]

M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003. (See http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.)

[SOAP/UDP]

H. Combs, et al, "SOAP-over-UDP," September 2004. (See http://schemas.xmlsoap.org/ws/2004/09/soap-over-udp.)

[UUID]

P. Leach, et al, "A UUID URN Namespace," December 2004. (See http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-05.txt.)

[WS-Addressing]

D. Box, et al, "Web Services Addressing (WS-Addressing)," August 2004. (See http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/.)

[WS-SecureConversation]

S. Anderson, et al, "Web Services Secure Conversation Language (WS-SecureConversation)," February 2005. (See http://schemas.xmlsoap.org/ws/2005/02/sc.)

[WS-Trust]

S. Anderson, et al, "Web Services Trust Language (WS-Trust)," February 2005. (See http://schemas.xmlsoap.org/ws/2005/02/trust.)

[WSDL 1.1]

E. Christensen, et al, "Web Services Description Language (WSDL) 1.1," March 2001. (See http://www.w3.org/TR/2001/NOTE-wsdl-20010315.)

[WS-Security]

A. Nadalin, et al, "Web Services Security: SOAP Message Security," March 2004. (See http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.)

[XML Schema, Part 1]

H. Thompson, et al, "XML Schema Part 1: Structures," May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.)

[XML Schema, Part 2]

P. Biron, et al, "XML Schema Part 2: Datatypes," May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.)

[XML Sig]

D. Eastlake, et al, "XML-Signature Syntax and Processing," February 2002. (See http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/.)

Appendix I – Application Sequencing

The Application Sequencing header block allows a receiver to order messages that contain this header block though they might have been received out of order. It is used by this specification to allow ordering messages from a Target Service; it is also expected that this header block will be useful in other applications.

The normative outline for the application sequence header block is:

```
<s:Envelope ...>
  <s:Header ...>
    <d:AppSequence InstanceId="xs:unsignedInt"
                  [SequenceId="xs:anyURI"]?
                   MessageNumber="xs:unsignedInt"
                   ... />
  </s:Header>
  <s:Body ...> ... </s:Body>
</s:Envelope>
```

The following describes normative constraints on the outline listed above:

/s:Envelope/s:Header/d:AppSequence/@InstanceId

MUST be incremented by >= 1 each time the service has gone down, lost state, and came back up again. SHOULD NOT be incremented otherwise. Means to set this value include, but are not limited to:

- A counter that is incremented on each 'cold' boot
- The boot time of the service, expressed as seconds elapsed since midnight January 1, 1970

/s:Envelope/s:Header/d:AppSequence/@SequenceId

Identifies a sequence within the context of an instance identifier. If omitted, implied value is the null sequence. MUST be unique within ./@InstanceId.

/s:Envelope/s:Header/d:AppSequence/@MessageNumber

Identifies a message within the context of a sequence identifier and an instance identifier. MUST be incremented by >= 1 for each message sent. Transport-level retransmission MUST preserve this value.

Other components of the outline above are not further constrained by this specification.

Appendix II – XML Schema

A normative copy of the XML Schema [XML Schema Part 1, Part 2] description for this specification can be retrieved from the following address:

http://schemas.xmlsoap.org/ws/2005/04/discovery/ws-discovery.xsd

A non-normative copy of the XML Schema description is listed below for convenience.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    targetNamespace="http://schemas.xmlsoap.org/ws/2005/04/discovery"
    xmlns:tns="http://schemas.xmlsoap.org/ws/2005/04/discovery"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    blockDefault="#all" >

  <xs:import
      namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
      schemaLocation="http://schemas.xmlsoap.org/ws/2004/08/addressing"
/>

  <!-- ////////////////////// Discovery Messages ////////////////////// -->

  <xs:element name="Hello" type="tns:HelloType" />
  <xs:complexType name="HelloType" >
    <xs:sequence>
      <xs:element ref="wsa:EndpointReference" />
      <xs:element ref="tns:Types" minOccurs="0" />
      <xs:element ref="tns:Scopes" minOccurs="0" />
      <xs:element ref="tns:XAddrs" minOccurs="0" />
      <xs:element ref="tns:MetadataVersion" />
      <xs:any namespace="##other"
              processContents="lax"
              minOccurs="0"
```

```
            maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<xs:simpleType name="RelationshipType" >
    <xs:restriction base="xs:QName" >
      <xs:enumeration value="tns:Suppression" />
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="OpenRelationshipType" >
  <xs:union memberTypes="tns:RelationshipType xs:QName" />
</xs:simpleType>

<xs:element name="Bye" type="tns:ByeType" />
<xs:complexType name="ByeType" >
  <xs:sequence>
    <xs:element ref="wsa:EndpointReference" />
    <xs:element ref="tns:Types" minOccurs="0" />
    <xs:element ref="tns:Scopes" minOccurs="0" />
    <xs:element ref="tns:XAddrs" minOccurs="0" />
    <xs:element ref="tns:MetadataVersion" minOccurs="0" />
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<xs:element name="Probe" type="tns:ProbeType" />
<xs:complexType name="ProbeType" >
  <xs:sequence>
    <xs:element ref="tns:Types" minOccurs="0" />
    <xs:element ref="tns:Scopes" minOccurs="0" />
```

```
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<xs:element name="ProbeMatches" type="tns:ProbeMatchesType" />
<xs:complexType name="ProbeMatchesType" >
  <xs:sequence>
    <xs:element name="ProbeMatch"
                type="tns:ProbeMatchType"
                minOccurs="0"
                maxOccurs="unbounded" >
    </xs:element>
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>
<xs:complexType name="ProbeMatchType" >
  <xs:sequence>
    <xs:element ref="wsa:EndpointReference" />
    <xs:element ref="tns:Types" minOccurs="0" />
    <xs:element ref="tns:Scopes" minOccurs="0" />
    <xs:element ref="tns:XAddrs" minOccurs="0" />
    <xs:element ref="tns:MetadataVersion" />
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
```

```
</xs:complexType>

<xs:element name="Resolve" type="tns:ResolveType" />
<xs:complexType name="ResolveType" >
  <xs:sequence>
    <xs:element ref="wsa:EndpointReference" />
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<xs:element name="ResolveMatches" type="tns:ResolveMatchesType" />
<xs:complexType name="ResolveMatchesType" >
  <xs:sequence>
    <xs:element name="ResolveMatch"
                type="tns:ResolveMatchType"
                minOccurs="0" />
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>
<xs:complexType name="ResolveMatchType" >
  <xs:sequence>
    <xs:element ref="wsa:EndpointReference" />
    <xs:element ref="tns:Types" minOccurs="0" />
    <xs:element ref="tns:Scopes" minOccurs="0" />
    <xs:element ref="tns:XAddrs" />
    <xs:element ref="tns:MetadataVersion" />
    <xs:any namespace="##other"
            processContents="lax"
```

```
            minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<xs:element name="Types" type="tns:QNameListType" />
<xs:simpleType name="QNameListType" >
  <xs:list itemType="xs:QName" />
</xs:simpleType>

<xs:element name="Scopes" type="tns:ScopesType" />
<xs:complexType name="ScopesType" >
  <xs:simpleContent>
    <xs:extension base="tns:UriListType" >
      <xs:attribute name="MatchBy" type="xs:anyURI" />
      <xs:anyAttribute namespace="##other" processContents="lax" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element name="XAddrs" type="tns:UriListType" />
<xs:simpleType name="UriListType" >
  <xs:list itemType="xs:anyURI" />
</xs:simpleType>

<xs:element name="MetadataVersion" type="xs:unsignedInt" />

<!-- ////////////////////// Faults ////////////////////// -->

<xs:simpleType name="FaultCodeType" >
    <xs:restriction base="xs:QName" >
      <xs:enumeration value="tns:MatchingRuleNotSupported" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="FaultCodeOpenType" >
```

```
  <xs:union memberTypes="tns:FaultCodeType xs:QName" />
</xs:simpleType>

<xs:element name="SupportedMatchingRules" type="tns:UriListType" />

<!-- ////////////////////// Compact Signature ////////////////////// -->

<xs:attribute name="Id" type="xs:ID"/>

<xs:element name="Security" type="tns:SecurityType" />
<xs:complexType name="SecurityType" >
  <xs:sequence>
    <xs:element ref="tns:Sig" minOccurs="0" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<xs:element name="Sig" type="tns:SigType" />
<xs:complexType name="SigType" >
  <xs:sequence>
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="Scheme" type="xs:anyURI" use="required" />
  <xs:attribute name="KeyId" type="xs:base64Binary" />
  <xs:attribute name="Refs" type="xs:IDREFS" use="required" />
  <xs:attribute name="Sig" type="xs:base64Binary" use="required" />
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<!-- ////////////////////// General Headers ////////////////////// -->

<xs:element name="AppSequence" type="tns:AppSequenceType" />
<xs:complexType name="AppSequenceType" >
```

```
    <xs:complexContent>
      <xs:restriction base="xs:anyType" >
        <xs:attribute name="InstanceId"
                      type="xs:unsignedInt"
                      use="required" />
        <xs:attribute name="SequenceId" type="xs:anyURI" />
        <xs:attribute name="MessageNumber"
                      type="xs:unsignedInt"
                      use="required" />
        <xs:anyAttribute namespace="##other" processContents="lax" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

</xs:schema>
```

Appendix III – WSDL

A normative copy of the WSDL [WSDL 1.1] description for this specification can be retrieved from the following address:

http://schemas.xmlsoap.org/ws/2005/04/discovery/ws-discovery.wsdl

A non-normative copy of the WSDL description is listed below for convenience.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    targetNamespace="http://schemas.xmlsoap.org/ws/2005/04/discovery"
    xmlns:tns="http://schemas.xmlsoap.org/ws/2005/04/discovery"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" >

  <wsdl:types>
    <xs:schema>
      <xs:import
          namespace="http://schemas.xmlsoap.org/ws/2005/04/discovery"
    schemaLocation
    ="http://schemas.xmlsoap.org/ws/2005/04/discovery/ws-discovery.xsd"
          />
```

```
  </xs:schema>
</wsdl:types>

<wsdl:message name="HelloMsg" >
  <wsdl:part name="body" element="tns:Hello" />
</wsdl:message>

<wsdl:message name="ByeMsg" >
  <wsdl:part name="body" element="tns:Bye" />
</wsdl:message>

<wsdl:message name="ProbeMsg" >
  <wsdl:part name="body" element="tns:Probe" />
</wsdl:message>

<wsdl:message name="ProbeMatchMsg" >
  <wsdl:part name="body" element="tns:ProbeMatches" />
</wsdl:message>

<wsdl:message name="ResolveMsg" >
  <wsdl:part name="body" element="tns:Resolve" />
</wsdl:message>

<wsdl:message name="ResolveMatchMsg" >
  <wsdl:part name="body" element="tns:ResolveMatches" />
</wsdl:message>

<wsdl:portType name="TargetService" >
  <wsdl:operation name="HelloOp" >
    <wsdl:output message="tns:HelloMsg"
    wsa:Action
    ="http://schemas.xmlsoap.org/ws/2005/04/discovery/Hello"
    />
  </wsdl:operation>
  <wsdl:operation name="ByeOp" >
    <wsdl:output message="tns:ByeMsg"
```

```
    wsa:Action
    ="http://schemas.xmlsoap.org/ws/2005/04/discovery/Bye"
    />
  </wsdl:operation>
  <wsdl:operation name="ProbeOp" >
    <wsdl:input message="tns:ProbeMsg"
    wsa:Action
    ="http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe"
    />
  </wsdl:operation>
  <wsdl:operation name="ProbeMatchOp" >
    <wsdl:output message="tns:ProbeMatchMsg"
    wsa:Action
    ="http://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches"
    />
  </wsdl:operation>
  <wsdl:operation name="ResolveOp" >
    <wsdl:input message="tns:ResolveMsg"
    wsa:Action
    ="http://schemas.xmlsoap.org/ws/2005/04/discovery/Resolve"
    />
  </wsdl:operation>
  <wsdl:operation name="ResolveMatchOp" >
    <wsdl:output message="tns:ResolveMatchMsg"
    wsa:Action
    ="http://schemas.xmlsoap.org/ws/2005/04/discovery/ResolveMatches"
    />
  </wsdl:operation>
</wsdl:portType>

<!-- If this portType is included in EndpointReference/Types, it
     indicates the Target Service is a Discovery Proxy. Discovery
     Proxies also implement tns:TargetService and optionally other
     message exchanges defined elsewhere.
-->
<wsdl:portType name="DiscoveryProxy" />
```

```
</wsdl:definitions>
```