XEP-0278: Jingle Relay Nodes

Thiago Camargo
mailto:thiago@xmppjingle.com
xmpp:barata7@gmail.com

2017-09-14
Version 0.3

Status	Type	Short Name
Experimental	Standards Track	jinglenodes

This documents specifies how Jingle Clients can interact with Jingle Relay Nodes Services and how XMPP entities can provide, search and list available Jingle Relay Nodes.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Jingle Nodes is an XMPP Based Relay Service providing standard UDP/TCP Relay, but negotiated via XMPP. Jingle Relay Nodes are intend to provide easy to use Jingle Relay Type Candidates that can be used in ICE-UDP, RAW-UDP, TCP Jingle Sessions. Relay Candidates can provide NAT Traversal for Jingle users with or without STUN/TURN Support. The main benefits of Jingle Relay Nodes is the easy to use candidates, Jingle Clients can became a Node and Jingle Relay Nodes are published via XMPP, meaning every Client or Server can also act as a tracker of another Nodes and STUN Servers.

2 Requirements

Jingle Relay Nodes MUST be binded directly to a Public IP address without firewall for traffic on the port range reserved to be used by relay candidates. This is the main and unique requirement for a peer provide Relay Nodes Service. All signalling, request, response and publishing is done via XMPP, not requiring any extra stack or protocol in the Client or Server, for usage and discoveral of Nodes.

3 Terminology

3.1 Glossary

Relay Relays are mainly used to transfer traffic to servers located on a NAT:ed (masqueraded) network, where the IP addresses on the NAT:ed network cannot be accessed from the outside. When you use an IP address that is for local use only, you must use NAT and relays because these IP addresses cannot be accessed in any other way. Relays can also be used for non-NAT:ed networks.

Jingle Relay Node Is an instance of a Relay Service that is negotiable via XMPP, following the procedures described on this Extension.

Tracker Is the entity that trackers Jingle Relay Nodes and also publishes the list upon request. Potentially all Jingle Clients might act as a Node Tracker.

Channel Is the UDP/TCP Relay Channel, provided by the a Jingle Relay Node. The channel act as a NAT Traversal Channel in order to delivery and receive media.

Requester Is the Jingle Client that makes requests and make use of a Channel. The Requester receives a relay Transport Candidate that can be used with Jingle ICE-UDP Transport Method (XEP-0176) XEP-0176: Jingle ICE-UDP Transport Method <https://xmpp.org/extensions/xep-0176.html>. or Jingle Raw UDP Transport Method (XEP-0177) XEP-0177: Jingle Raw UDP Transport Method <https://xmpp.org/extensions/xep-0177.html>..

3.2 Conventions

In diagrams, the following conventions are used:

- Single-dashed lines (---) represent Jingle stanzas that are sent via the XMPP signalling channel.
- Double-dashed lines (===) represent media packets that are sent via the data channel, which typically is not an XMPP channel (although the Jingle In-Band Bytestreams Transport Method is an exception) but instead is a direct or mediated channel between the endpoints.

4 Flow Example

After the Jingle Clients gets succesfully connected to the XMPP Server, it MAY want to start discoverying available Relay Services in order to cache some entries. Having cached Relay Service Addresses is recommended as it speeds up the session setup time as the Client don't need to search for available Relay Services right before a session is started or received.

4.1 Jingle Client Checks for Tracker or Relay Services on its own Server Domain

A Jingle Client MAY start the search for Relay Services by querying his own XMPP Server Domain.
Note: This is a good implementation practice.

Listing 1: Service List Request

```
<iq from='romeo@montague.lit/orchard'
    id='uw72g176'
    to='montague.lit'
    type='get'>
  <services xmlns='http://jabber.org/protocol/jinglenodes'/>
</iq>
```

Listing 2: Tracker Returned Known Public Relay Services

```
<iq from='montague.lit'
    id='uw72g176'
    to='romeo@montague.lit/orchard'
    type='result'>
  <services xmlns='http://jabber.org/protocol/jinglenodes'>
    <relay policy='public' address='montague.lit' protocol='udp'/>
    <tracker policy='public' address='capulet.lit' protocol='udp'/>
    <turn policy='public' address='stun.capulet.lit' protocol='udp'/>
```

```
    <stun policy='public' address='200.111.111.111' port='3857'
       protocol='udp'/>
  </services>
</iq>
```

In this example 'montague.lit' XMPP Domain a Relay Service and a Tracker Service. The Relay Service can be contacted in order to retrieve Relay Channels. The Tracker Service can be contacted in order to retrieve its known services.

4.2 Jingle Client Searches for Services on a Retrieved Tracker Service

A Jingle Client MAY NOT be satisfied with only one Relay Service entry found. So it keeps the search on the known Tracker Services.

Listing 3: Service List Request

```
<iq from='romeo@montague.lit/orchard'
    id='uw72g177'
    to='capulet.lit'
    type='get'>
  <services xmlns='http://jabber.org/protocol/jinglenodes'/>
</iq>
```

Listing 4: Tracker Returned Known Public Relay Services

```
<iq from='capulet.lit'
    id='uw72g177'
    to='romeo@montague.lit/orchard'
    type='result'>
  <services xmlns='http://jabber.org/protocol/jinglenodes'/>
</iq>
```

In this example 'capulet.lit' returned an empty service list, meaning that it does NOT known ANY Relay or Tracker Services.

4.3 Jingle Client Searches for Services on online Roster Entries

A Jingle Client MAY NOT be satisfied with only one Relay Service entry found. So it keeps the search on his Roster Items until find the desired amount of Relay Services, or while it does NOT exceed a search depth or ANY other Client implementation policy. The Client SHOULD keep a list of visited Tracker Services in order to avoid searching twice in same Service Entity.

Listing 5: Service List Request

```
<iq from='romeo@montague.lit/orchard'
```

```
    id='uw72g177'
    to='juliet@capulet.lit/balcony'
    type='get'>
  <services xmlns='http://jabber.org/protocol/jinglenodes'/>
</iq>
```

Listing 6: Tracker Returned Known Public Relay Services

```
<iq from='juliet@capulet.lit/balcony'
    id='uw72g177'
    to='romeo@montague.lit/orchard'
    type='result'>
  <services xmlns='http://jabber.org/protocol/jinglenodes'>
    <relay policy='roster' address='juliet@capulet.lit/balcony'
       protocol='udp'/>
  </services>
</iq>
```

In this example 'juliet@capulet.lit/balcony' returned a Relay Service entry that is restricted to its roster. This Service is usable as the requester has 'juliet@capulet.lit/balcony' on its roster. Although, services with policy 'roster' MUST NOT be listed in Tracker Responses expects in Tracker Responses that comes from the Service Entity itself, in this case 'juliet@capulet.lit/balcony'.
In the presented example 'romeo@montague.lit/orchard' knows that 'juliet@capulet.lit/balcony' provides Relay Service, but if another entity requests 'romeo@montague.lit/orchard' its known services, it MUST NOT include 'juliet@capulet.lit/balcony' as it is a roster restricted entry.

4.4 Jingle Client Consuming the Relay Service

A Jingle Client with Internet connectivity wheter with direct access to a public IP or not, can potentially provide the Relay Service becaming itself a Jingle Relay Node. The service can intend to provide a public service, or a restricted services based on user preferences, like buddylist, whitelist, blacklist, domain, etc...
Note: It is NOT mandatory to became a Jingle Relay Node it is OPTIONAL and SHOULD be done ONLY under user awareness and concentiment.

Listing 7: UDP Relay Channel request

```
<iq from='romeo@montague.lit/orchard'
    id='uw72g176'
    to='juliet@capulet.lit/balcony'
    type='get'>
  <channel xmlns='http://jabber.org/protocol/jinglenodes#channel'
     protocol='udp'/>
</iq>
```

Listing 8: Candidate Returned by a Node with channel bandwidth throttle

```
<iq from='juliet@capulet.lit/balcony'
    id='uw72g176'
    to='romeo@montague.lit/orchard'
    type='result'>
  <channel component='1'
      id='el0747fg11'
      host='200.20.2.10'
      localport='35800'
      remoteport='35802'
      protocol='udp'
      maxkbps='120'
      expire='60'/>
</iq>
```

After receiving the <channel/> the requester MUST send his stream to 'host' and 'localport' pair and send a <candidate/> containing the 'host' and 'remoteport' values.

4.5 Jingle Client Consuming TURN Credentials Service

A Jingle Client can request volative TURN credentials, to be used in cases where connectivity check is a requirement. Like, for example, WebRTC. The concept and mechanism is quite similar to the RFC draft REST API For Access To TURN Services'.
TURN provides an access control mechanism described in RFC 5389 [1], where long-term credentials are provided as part of the TURN protocol. Therefore the credentials provided in this Jingle Nodes mechanism are time-limited, but SHOULD be used as long-term credentials, when authentication against a TURN Server.
Note: There is no need to run TURN server or support within a Jingle Relay. This mechanism allows decoupled deployment of distributed TURN Servers, without the requirement of database based authentication.

Listing 9: TURN Credentials request

```
<iq from='romeo@montague.lit/orchard'
    id='uw72g176'
    to='juliet@capulet.lit/balcony'
    type='get'>
  <turn xmlns='http://jabber.org/protocol/jinglenodes#turncredentials'
      protocol='udp'/>
</iq>
```

Listing 10: TUNR Credentials Returned by the service

```
<iq from='juliet@capulet.lit/balcony'
    id='uw72g176'
```

[1] RFC 5389: Session Traversal Utilities for NAT (STUN) <http://tools.ietf.org/html/rfc5389>.

```
    to='romeo@montague.lit/orchard'
    type='result'>
  <turn ttl='60000'
      uri='turn:200.20.2.10:1984?transport=udp'
      username='1433895918506:romeocapulet'
      password='1Dj9XZ5fwvKS6YoQZOoORcFnXaI='
      />
</iq>
```

5 Services Definitions

5.1 Relay Channel Service

A Relay Channel Service is responsible for providing the actual Relay Services. It will receive Channel Requests, allocate the Relay Channel and return the ready to use details. If a Jingle Client knows the service address of one valid and reliable Relay Service, that is enough for place and receive Jingle Calls and transmit both ways media streams based on UDP.
Relay Channel Services support can be discovered by searching using Tracker Services. It can also be discovered by service discovery described in this document.
Note: Jingle Relay Channels can be used with RAW-UDP and ICE-UDP Jingle Transports.

5.2 Tracker Service

A Tracker Service is responsible for providing addresses of known Relay Channel Services and other Tracker Services as well.

5.2.1 Services Types

Tracker entries MUST contain a 'type' attribute that represents the usage policy according to the table below:

Type	Definition
public	Relays Services that are meant and opened for public usage, SHOULD use the type 'public'. Meaning that every user can make use of its services. This type SHOULD be published by Tracker Services.
roster	Relay Services that only provides Channels for users that are in it own roster, SHOULD use the type 'roster'. Meaning that only presence subscribed buddies can make use of its service. Common usage is XMPP Clients with Relay Services Capabilities. This type SHOULD NOT be published by Tracker Services as if it is roster only, the requester SHOULD have the entity already added to his roster, which also mean that it SHOULD be discoverable on roster level.

6 Formal Definition

6.1 Channel Element

The <channel/> element MUST be empty.
The attributes of the <channel/> element are as follows.

Attribute	Definition	Inclusion
id	A random candidate identifier generated by the Relay Service, which effectively maps to the created Channel; this SHOULD match the XML Nmtoken production See <http://www.w3.org/TR/2000/WD-xml-2e-20000814#NT-Nmtoken> so that XML character escaping is not needed for characters such as '&'. In some situations the Jingle session identifier might have security implications. See RFC 4086 RFC 4086: Randomness Requirements for Security <http://tools.ietf.org/html/rfc4086>. regarding requirements for randomness.	REQUIRED on response, NOT RECOMMENDED on requests
host	The IP address or Host address of the Relay Channel.	REQUIRED on response
localport	The port number to be used by the channel requester.	REQUIRED on response
remoteport	The port number to be offered to the remote party.	REQUIRED on response
protocol	The protocol supported by the retrieved channel.	REQUIRED on response
maxkbps	The maximum bandwidth supported by the channel.	OPTIONAL on response, NOT RECOMMENDED on requests.
expire	The maximum amount of seconds that the channel can stay without receiving packets, without being deactivated and closed.	REQUIRED

6.1.1 Host Attribute

The value of the 'host' attribute MUST be one IP address or a DNS resolvable address. That is the address to be used on candidate offering and also the IP to be used when sending out the media traffic.

6.1.2 Localport Attribute

The value of the 'localport' attribute MUST be a valid IP Port number. This port MUST be used as the media traffic destination port of the channel requester. Session Initiator and responder MUST NOT offer this port.
The rule is simple and unique, the requester of the channel MUST send the media streams to 'localport' and use 'remoteport' in the 'candidate' element to be offered in the Jingle Session.
For transparent compatibility with major RTP Proxy Deployments, an RCTP Port is allocated and defined by default at Localport Attribute Value plus one. (Localport + 1)

6.1.3 Remoteport Attribute

The value of the 'remoteport' attribute MUST be a valid IP Port number. This port MUST be used as media traffic destination port of the other party. Channel requester MUST use this port value in the candidate offer in combination with the 'host' attribute. Channel requester MUST NOT send any media stream to this port.
For transparent compatibility with major RTP Proxy Deployments, an RCTP Port is allocated and defined by default at Remoteport Attribute Value plus one. (Localport + 1)

6.1.4 Protocol Attribute

The value of the 'protocol' attribute MUST be a valid protocol value: 'udp' or 'tcp' as also defined in the XML Schema

6.1.5 Maxkbps Attribute

The value of the 'maxkbps' attribute MUST be a valid integer value representing the maximum kilobits per seconds the channel supports. This attribute is optional and MAY be used in Relay Channel with bandwidth limitation.

6.1.6 Expire Attribute

The value of the 'expire' attribute MUST be a valid integer value representing the maximum seconds that the channel can stay without receiving any traffic without being deactivated and

closed. This attribute is required and SHOULD be used in all Relay Channels.

6.2 Services Element

The <services/> element MAY be empty or contain <relay/>, <stun/> and/or <tracker/> elements.
The attributes of the <relay/> and <tracker/> element are as follows.

Attribute	Definition	Inclusion
policy	The policy of the service. If the service is public, MUST be 'public' if it is restricted to roster, MUST be 'roster'.	REQUIRED
address	For Relay and Tracker Services the JID. For STUN Service the IP or Host address.	REQUIRED
protocol	The protocol supported by the retrieved service.	REQUIRED
port	The port number of the STUN service. This field is only used in STUN Service entries.	REQUIRED for STUN entries

6.3 TURN Credentials Service Element

The attributes of the <turn/> element are as follows.

Attribute	Definition	Inclusion
ttl	The duration in seconds for which the provided credentials are valid.	REQUIRED
uri	The TURN Server URI.	REQUIRED
username	The username to be used on TURN authentication. THe recommended format is a colon-delimited concatenation of expiration timestamp and the requester bare JID.	REQUIRED
password	The ppassword to be used on TURN authentication. Is the result of 'base64(hmac(secret_key, username))'. Where 'secret_key' is shared between the TURN server and entity providing the credentials.	REQUIRED

6.3.1 Ttl Attribute

The duration in seconds for which the provided credentials are valid. The usual and recommended value is 86400 seconds (one day).

6.3.2 URI Attribute

The TURN Server URI as described in I-D.petithuguenin-behave-turn-uris

6.3.3 Username Attribute

WebRTC's TURN request uses the 'username' value for its USERNAME and PASSWORD attributes, for the input to the MESSAGE-INTEGRITY hash.

6.3.4 Password Attribute

Along with 'username', WebRTC's TURN request uses the 'password' value for its USERNAME and PASSWORD attributes, for the input to the MESSAGE-INTEGRITY hash.

7 Determining Support

To advertise its support for the Jingle Nodes support, when replying to Service Discovery (XEP-0030) [2] information requests an entity MUST return URNs for any version of this protocol that the entity supports -- e.g., "http://jabber.org/protocol/jinglenodes" for this version(see Namespace Versioning regarding the possibility of incrementing the version number).
If the entity supports, Jingle Nodes as a Tracker, it MUST reply to Service Discovery (XEP-0030) [3] with "http://jabber.org/protocol/jinglenodes". If it also provides the Jingle Nodes Relay Services, it MUST reply with the URN "http://jabber.org/protocol/jinglenodes#channel".
For optimization purpose the Client SHOULD check for Jingle Nodes support based on entity presence capabilities Entity Capabilities (XEP-0115) [4], which SHOULD contain the keywork "jn-v0".

Listing 11: Service discovery information request

```
<iq from='romeo@montague.lit/orchard'
```

[2] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[3] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[4] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

```
    id='uw72g176'
    to='juliet@capulet.lit/balcony'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 12: Service discovery information response

```
<iq from='juliet@capulet.lit/balcony'
    id='uw72g176'
    to='romeo@montague.lit/orchard'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
      <feature var='http://jabber.org/protocol/jinglenodes'/>
      <feature var='http://jabber.org/protocol/jinglenodes#channel'/>
      <feature var='http://jabber.org/protocol/jinglenodes#
          turncredentials'/>
  </query>
</iq>
```

In order for an application to determine whether an entity supports this protocol, where possible it SHOULD use the dynamic, presence-based profile of service discovery defined in Entity Capabilities (XEP-0115) [5]. However, if an application has not received entity capabilities information from an entity, it SHOULD use explicit service discovery instead.

8 Recommended Use Cases

8.1 Jingle Client with RAW-UDP Transport without any NAT detection mechanism

A Jingle Client with only RAW-UDP support and any NAT detection mechanism can make use of Jingle Nodes. Although the traversal is not guarantee due problem with NAT configuration or firewalls, this method is fairly efficient as most SIP legacy services still provides their services in a similar way, meaning that this has the exactly same accertivity and reliability than regular SIP Services. This method is extremelly useful specially for simple platforms like mobile and clients in early stage of development that still wants to offer voice and video support.
Note: This use case is also similar to a Jingle to SIP Interoperability Service.

8.2 Jingle Client that uses WebRTC with TURN required

A Jingle Client that uses WebRTC, therefore requiring a TURN Server and its credentials to successfully alocate channels. This specification describes a simple way of discovering TURN

[5] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

Services and retrieving credentials to successfully allocate channels. This also simplifies deployment and distribution of TURN servers, since its stateless authentication does not require connectivity to database authoriztion services.

8.3 Jingle Client with ICE-UDP Transport with STUN support but no TURN support

A Jingle Client with STUN support but no TURN support can use Relay Node Services as the fallback candidate instead of a TURN candidate. For instance, after a connectivity check proccess, none of the direct candidates worked. The Client can use the Relay Node Candidate as the fallback candidate(the lowest priority candidate).

8.4 Jingle Client with ICE-UDP Transport with STUN and TURN support

A Jingle Client with STUN and TURN support still might need a Relay Candidate, specially as TURN servers are not widely deployed and don't have any mechanism to publish available services for users. TURN servers also requires specific UDP traffic in specific port ranges which can be blocked by users networks. In this case is very useful to have an extra fallback candidate that can be negotiated via XMPP rather than over UDP.
Note: Jingle Relay Nodes Services are much more likely to be found and be available than TURN servers, as users can also provide Jingle Relay Nodes services themselves or provide a list of known and available services, in this last case behaving a tracker. Is very likely that Jingle Relay Nodes will be available and discovered more easily and often than TURN servers.

8.5 XMPP Client or Component in a Public IP

A XMPP Client or Component with direct access to a public IP can potentially provide the UDP Relay Service becaming itself a Jingle Relay Node. The service can intend to provide a public service, or a restricted services based on user preferences, like buddylist, whitelist, blacklist, domain, etc...
Note: It is NOT mandatory to became a Jingle Relay Node. This is OPTIONAL and SHOULD be done with user awareness and concentiment.

9 Implementation Notes

When using a candidate provided by a Jingle Relay Service, the Jingle Client MUST set the candidate attribute to 'relay'. In order to make sure the other client won't allocate another channel as well. Which would lead in audio connectivity issues. In brief, if the caller is using a Relay Candidate the calle MUST NOT use another Relay Candidate discovered by itself. If a caller is not using a Relay Candidate(which can be determined by the candidate 'type'

attribute) the callee MAY use a Relay Candidate in order to ensure communication.

10 Security Considerations

Relay Channels auto expires MUST expire on traffic inactivity. The inactivity timeout recommended is 60 seconds.
It is heavily recommended that the Super Node implements throttle:

- Based on JID, allowing the control of how many concurrent channels an specific JID can have.
- Based on JID, allowing the control of how many channel requests an specific JID can request in a time period.
- Based on Bandwidth, allowing the control of how much bandwidth a channel can use. The maximum bandwidth SHOULD be included on the candidate element provided by a Super Node on the attribute maxkbps. If no attribute is present, it means that it has no bandwidth control.

Listing 13: Channel Returned by a Node with bandwidth throttle (stub)

```
<channel component='1'
    id='el0747fg11'
    host='200.20.2.10'
    localport='35800'
    remoteport='35802'
    protocol='udp'
    maxkbps='120'/>
```

11 XML Schema

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://jabber.org/protocol/jinglenodes'
    xmlns='http://jabber.org/protocol/jinglenodes'
    elementFormDefault='qualified'>

  <xs:element name='channel'>
        <xs:attribute name='id' type='xs:string' use='required'/>
        <xs:attribute name='host' type='xs:string' use='required'/>
        <xs:attribute name='localport' type='xs:string' use='required'
            />
```

```
      <xs:attribute name='remoteport' type='xs:string' use='required
         '/>
<xs:attribute name='protocol' use='required'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='udp'/>
            <xs:enumeration value='tcp'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name='maxkbps' type='xs:string' use='optional'/>
      <xs:attribute name='expire' type='xs:string' use='required'/>
</xs:element>

<xs:element name='turn'>
      <xs:attribute name='ttl' type='xs:string' use='required'/>
      <xs:attribute name='uri' type='xs:string' use='required'/>
      <xs:attribute name='username' type='xs:string' use='required'/
         >
      <xs:attribute name='password' type='xs:string' use='required'/
         >
</xs:element>

<xs:element name='services'>
  <xs:complexType>
    <xs:sequence>
      <xs:element name='relay'
                  type='serviceElementType'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
      <xs:element name='tracker'
                  type='serviceElementType'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
      <xs:element name='stun'
                  type='serviceElementType'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
      <xs:element name='turn'
                  type='serviceElementType'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name='serviceElementType'>
  <xs:simpleContent>
    <xs:extension base='empty'>
```

```
        <xs:attribute name='address' type='xs:string' use='required'/>
        <xs:attribute name='port' type='xs:string' use='optional'/>
        <xs:attribute name='policy' use='required'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='public'/>
              <xs:enumeration value='roster'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name='protocol' use='required'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='udp'/>
              <xs:enumeration value='tcp'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```