

[JET-PROTO]: JET Relay Protocol

Revision History

Revision summary			
Author	Date	Revision history	Comments
Marc-André Moreau	10/25/2018	0.1	Initial draft
Marc-André Moreau	07/17/2019	0.2	Major update

Contents

1	Introduction	3
1.1	Glossary	3
1.2	References	3
1.2.1	Normative References	3
1.3	Overview	3
1.4	Prerequisites/Preconditions	3
1.5	Applicability Statement	3
2	Messages	4
2.1	Transport	4
2.2	Message Syntax	4
2.2.1	Protocol Messages	4
2.2.1.1	JET_PACKET	4
2.2.2	REST API	4
2.2.2.1	Jet Association Creation	5
2.2.2.2	Jet Candidate Format	7
2.2.2.2.1	URL Format	7
2.2.3	Binary Protocol	9
2.2.3.1	TCP Relay	Error! Bookmark not defined.
2.2.3.1.1	TCP Accept	9
2.2.3.1.2	TCP Connect	9
2.2.3.2	TCP Direct	Error! Bookmark not defined.
2.2.3.2.1	TCP Connect	Error! Bookmark not defined.
2.2.3.3	TLS Relay	Error! Bookmark not defined.
2.2.3.4	TLS Direct	Error! Bookmark not defined.
2.2.3.5	TCP Relay (Old)	Error! Bookmark not defined.
2.2.3.5.1	TCP Accept	Error! Bookmark not defined.
2.2.3.5.2	TCP Connect	Error! Bookmark not defined.
2.2.4	WebSocket Protocol	10
2.2.4.1	WebSocket Server Accept	11
2.2.4.2	WebSocket Client Connect	11
2.3	Protocol details	13
2.3.1	Abstract Data Model	13
2.3.1.1	Association	13
2.3.1.2	Candidate	13
2.3.1.3	Usable Candidate	13
2.3.1.4	Selected Candidate	13
2.3.1.5	Signaling Server	13
2.3.2	Processing and Sequencing Rules	13
2.3.2.1	Connection Initiation	13
2.3.2.2	Candidate Gathering	14
2.3.2.2.1	Relay Candidate Gathering	14
2.3.2.2.2	Direct Candidate Gathering	14

1 Introduction

This document specifies the JET Relay Protocol.

1.1 Glossary

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>.

1.3 Overview

The JET protocol bears some similarities with the SOCKS proxy protocol and the routing token packets often used in remote desktop connections for load balancing and session selection. However, all of these protocols make connections in a forward manner: the client connects to the proxy, then the proxy connects to the server and then relays the traffic. The JET protocol is designed to relay TCP traffic between a TCP client and server using only outgoing TCP connections.

1.4 Prerequisites/Preconditions

The JET protocol requires a TCP transport.

1.5 Applicability Statement

The JET protocol is suitable for simple, efficient relaying TCP, TLS or WebSocket traffic.

2 Messages

2.1 Transport

The JET protocol is designed to provide a simple, efficient way to relay TCP traffic between two nodes that can only perform outgoing TCP connections to the same server, using a rendezvous connection style. Alternatively, a JET packet can be sent between a client and server as a way to discover a direct route.

2.2 Message Syntax

The following sections specify the JET protocol message syntax. All fields defined in this document use big endian byte ordering and are byte-aligned to their sizes (a field of 4 bytes starts at an offset that is a multiple of 4).

2.2.1 Protocol Messages

All JET binary protocol messages sent over TCP or TLS are contained within a JET_PACKET structure. The JET WebSocket protocol makes use of HTTP requests and the WebSocket handshake request path to achieve the same goal.

2.2.1.1 JET_PACKET

The JET_PACKET structure contains a routing token used to associate two indirect TCP connections.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	1	2	3	4	5	6	7	8	9	30	1
signature																															
size																flags								mask							
payload (variable)																															
...																															

signature (4 bytes): The packet signature. MUST be set to { 0x4A, 0x45, 0x54, 0x00 } ("JET").

size (2 bytes): The total size of the packet, including headers. The minimum size is 8.

flags (1 byte): This field is reserved for future use and MUST be set to zero.

mask (1 byte): This field contains a one-byte mask that MUST be applied to the payload.

payload (variable): This field contains a masked HTTP request or response using the value from the mask field.

2.2.2 REST API

The REST API is used to create Jet associations and optionally enforce authentication. It is optional for the JET binary protocol, but it is required for the JET WebSocket protocol.

2.2.2.1 Jet Association

The JET WebSocket protocol requires that the association be created with an HTTP request/response. While this call could be made by a backend server, it is recommended to make the call from endpoint that will act as a server, such that DNS load balancing can be implemented properly by selecting the closest Jet instance.

An association associates a client and a server independently from the underlying transport used. When a new association is created, a list of candidates is produced and sent to both peers to be tested for connectivity. A candidate is considered usable if both peers managed to connect using the same association id. Multiple usable candidates will normally be opened this way, but only one will be selected by the client, while all others will be closed without application traffic being sent.

The usage of an API key or authentication for this API endpoint is recommended but not required.

Jet association creation:

>> Request:

POST /jet/association/e6ec698c-5793-4c63-af79-bd644ccf022f HTTP/1.1

Host: jet.wayk.net

<< Response:

HTTP/1.1 200 OK

Jet association deletion:

>> Request:

DELETE /jet/association/e6ec698c-5793-4c63-af79-bd644ccf022f HTTP/1.1

Host: jet.wayk.net

<< Response:

HTTP/1.1 200 OK

Jet association info:

>> Request:

GET /jet/association/e6ec698c-5793-4c63-af79-bd644ccf022f HTTP/1.1

Host: jet.wayk.net

<< Response:
HTTP/1.1 200 OK

2.2.2.2 Jet Candidate Gathering

>> Request:
POST /jet/gather/e6ec698c-5793-4c63-af79-bd644ccf022f HTTP/1.1
Host: jet.wayk.net

<< Response:
HTTP/1.1 200 OK
{
 "id": "e6ec698c-5793-4c63-af79-bd644ccf022f",
 "candidates": [
 { "url": "tcp://jet101.wayk.net:8080",
 "id": "1ff84b5f-5a62-4124-bf61-381a5c55db89" },
 { "url": "tls://jet101.wayk.net:443",
 "id": "e9f4e1ea-0808-4e2a-8359-dd6e19c9148a" },
 { "url": "wss://jet101.wayk.net",
 "id": "b1cc4748-95a8-4064-9bd9-2e67b74a6fb9" }
]
}

The association id MUST be the same for all candidates, but each candidate MUST have a different id. A candidate id only needs to be unique within the same association. The association id can be the same across multiple distinct relay servers, as long as it is used to identify the same association.

If the signaling server policy allows direct connectivity (non-relay) between peers, this list of candidates will be enriched with direct connectivity candidates provided by the server endpoint. It is also possible to further enrich the list of relay candidates by using additional servers, such as a relay server available on the local network.

To avoid complicated management of global list of associations, each relay server SHOULD automatically delete inactive or unused ids after a defined period of time. An association is considered active on a given relay server if one of the candidates was selected, meaning there is a non-zero amount of bytes that were sent after the original JET handshake.

2.2.2.3 Jet Candidate Format

The JET candidate can be encoded in multiple ways, depending on how it is transmitted.

2.2.2.3.1 URL Format

A JET candidate is a URL representing a possible route that can be used for a connection.

If no scheme is specified, the "tcp://" scheme should be used. The default port can only be omitted for ws:// (80) and wss:// (443) schemes, otherwise it must be included in the URL.

tcp://jet101.wayk.net:8080

A TCP candidate using the JET binary protocol. This candidate is compatible with end-to-end TLS usage, where the JET exchange occurs prior to the TLS handshake.

tls://jet101.wayk.net:4343

A TLS candidate using the JET binary protocol. This candidate is not compatible with end-to-end TLS because the TLS handshake is performed with the Jet relay, followed by the JET exchange over TLS.

ws://jet101.wayk.net:8080

A WebSocket candidate without TLS. The WebSocket handshake request path contains the JET parameters.

wss://jet101.wayk.net:4343

A WebSocket candidate with TLS. The WebSocket handshake request path contains the JET parameters.

2.2.3 Message Protocol

The Jet message protocol is the equivalent of the Session Description Protocol (SDP) used in WebRTC. It uses a series of JSON-formatted messages sent over the signaling server to perform the negotiation of the Jet candidates leading up to the selection of a final candidate, after which the signaling server is no longer required.

id: jet association id

role: jet role ("client" or "server")

version: jet version (2 or 3)

transports : comma-separated list of supported transports (optional, offer message only)

candidates : list of jet candidates (answer message only)

candidate: selected candidate (complete message only)

2.2.3.1 Offer Message

The offer message is produced by the Jet client and contains relatively little information, as we expect the Jet server to perform most of the work.

```
{
  "id": "4daeb814-cdb6-4779-a16b-6479064e8107",
  "role": "client",
  "transports": "tcp,wss",
  "version": 2
}
```

The "transports" field was introduced in Jet V3, but since it is optional, it remains backwards compatible with Jet V2. The absence of the "transports" field means the server should not filter candidate transport types.

Since the offer message is always sent by the Jet client, the "role" field MUST be set to "client".

2.2.3.2 Answer Message

The answer message is produced by the Jet server as a response to the Jet client offer. It contains the host and relay candidates gathered by the server to be tested by the client.

```
{
  "id": "4daeb814-cdb6-4779-a16b-6479064e8107",
  "role": "server",
  "version": 2,
  "candidates": [
    { "id": "174a46de-7c56-30e0-e083-b6b03a2df15f",
      "url": "tcp://10.1.21.34:4489?ctype=host" },
    { "id": "fdc2726d-e228-59cd-3536-2c0919b509a3",
      "url": "tcp://192.168.25.1:4489?ctype=host" },
    { "id": "4fab2441-b8a5-4bd5-9831-fe7aa8cccb36",
      "url": "tcp://jet.wayk.net:8080?ctype=relay" },
    { "id": "b1b4db92-9cc0-49d6-a940-1846fc07cf05",
      "url": "wss://jet.wayk.net:7171?ctype=relay" }
  ]
}
```


If the client offer contained the "transports" field, the server SHOULD use it to filter out unsupported transport types from its list of candidates. This means that if "transports" is set to "wss", the server SHOULD return only "wss" candidates and avoid wasting time opening "tcp" candidates.

Since the answer message is always sent by the server, the "role" field MUST be set to "server".

2.2.3.3 Complete Message

The complete message is sent by the Jet client to the Jet server to tell which candidate it has selected, prompting all other unused tested candidates to be closed. Following this stage, only one candidate remains and becomes the actual transport used for the peer-to-peer session.

```
{"candidate":{"id":"4fab2441-b8a5-4bd5-9831-fe7aa8cccb36"}}
```

2.2.4 Binary Protocol

The JET binary protocol consists of a single request/response exchange at the beginning of the connection, either over TCP or over TLS. When done over TCP, a TLS handshake can follow. When done over TLS, the TLS handshake must be completed first, with the JET exchange done over TLS.

2.2.4.1 Binary Accept

>> Request:

GET /jet/accept/<association-id>/<candidate-id> HTTP/1.1

Host: jet101.wayk.net

Connection: Keep-Alive

Jet-Version: 2

<< Response:

HTTP/1.1 200 OK

Jet-Version: 2

After the response is sent by the server, the connection SHOULD remain open until a candidate is selected for the association.

2.2.4.2 Binary Connect

>> Request:

GET /jet/connect/<association-id>/<candidate-id> HTTP/1.1

Host: jet101.wayk.net
Connection: Keep-Alive
Jet-Version: 2

<< Response:
HTTP/1.1 200 OK
Jet-Version: 2

After the response is sent by the server, the connection SHOULD remain open until a candidate is selected for the association.

2.2.4.3 Binary Test

>> Request:
GET /jet/test/<association-id>/<candidate-id> HTTP/1.1
Host: jet101.wayk.net
Connection: Close
Jet-Version: 2

<< Response:
HTTP/1.1 200 OK
Jet-Version: 2

After the response is sent by the server, the connection MUST be closed by the server.

2.2.5 WebSocket Protocol

The JET WebSocket protocol consists of an HTTP request/response followed by a WebSocket handshake where the JET parameters are all encoded in the HTTP request path. The usage of the request path is necessary since it is not possible to add custom HTTP headers to a WebSocket handshake from a browser implementation.

2.2.5.1 WebSocket Accept

>> Request:

GET /jet/accept/<association-id>/<candidate-id> HTTP/1.1

Host: jet101.wayk.net

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Version: 13

Sec-WebSocket-Key: B0mBaSTOZiC2RAkp3ScC8MA/gaI=

<< Response:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: B0mBaSTOZiC2RAkp3ScC8MA/gaI=

2.2.5.2 WebSocket Connect

>> Request:

GET /jet/connect/<association-id>/<candidate-id> HTTP/1.1

Host: jet101.wayk.net

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Version: 13

Sec-WebSocket-Key: B0mBaSTOZiC2RAkp3ScC8MA/gaI=

<< Response:

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: B0mBaSTOZiC2RAkp3ScC8MA/gaI=

2.2.5.3 WebSocket Test

The WebSocket connection test opens a WebSocket connection that is closed immediately upon success. The goal of opening this WebSocket connection is to make sure the WebSocket connection upgrade works and is not broken in by traffic inspection proxies.

>> Request:

GET /jet/test/<association-id>/<candidate-id> HTTP/1.1

Host: jet101.wayk.net

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Version: 13

Sec-WebSocket-Key: B0mBaSTOZiC2RAkp3ScC8MA/gaI=

<< Response (success):

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: B0mBaSTOZiC2RAkp3ScC8MA/gaI=

No data should be sent over this WebSocket, and it should be closed immediately after it has been successfully opened by the server. The closing of the WebSocket by the server is expected, and should not be considered an error as long as the WebSocket could be opened with success.

<< Response (failure):

HTTP/1.1 404 Not Found

If the association id and candidate id encoded in the request path are not recognized, the server should simply return an HTTP 404 not found error. As for the client, it should consider anything other than the success case to be a candidate test failure.

2.3 Protocol details

The following section explain the abstract data model of the JET protocol, along with a detailed explanation of the sequence of events for different scenarios.

2.3.1 Abstract Data Model

This section provides definitions for abstract data model elements used in the JET protocol.

2.3.1.1 Association

An association uniquely represents the link between a client and a server, regardless of the underlying transport used. An association contains multiple candidates from which only one will be selected after a series of connectivity tests. Once a candidate is selected, the association is essentially the same as the selected candidate it contains, and the underlying transport can be passed to the application for usage as if it were a regular transport. An association is identified by a UUID string.

2.3.1.2 Candidate

A candidate is one of the possible transports that is part of an association. Candidate transports include TCP, TLS and WSS for both relay and direct connection types. To maximize chances of connectivity, multiple candidates are tested in parallel, creating a list of usable candidates from which one candidate deemed optimal will be selected. Usable candidates that do not get selected are simply disconnected. A candidate is identified by a UUID string, which needs to be unique only within the same association.

2.3.1.3 Usable Candidate

A usable candidate is a candidate for which the connectivity test was completed successfully. A connectivity test is considered successful if the client connect request was successful. Usable candidates are a subset of the total candidates.

2.3.1.4 Selected Candidate

The selected candidate is candidate chosen by the client from the usable candidates, based on adaptive logic. Even if the selected candidate SHOULD be the most optimal, a client can select any of the usable candidates.

2.3.1.5 Signaling Server

The signaling server is an external communication channel where two peers can exchange messages necessary to trigger the JET peer-to-peer connection process. The way this signaling server is implemented is outside the scope of the JET protocol.

2.3.2 Processing and Sequencing Rules

2.3.2.1 Connection Initiation

The connection initiation requires that both the client and server be connected have a usable external communication channel such as a signaling server. The client sends an SDP offer and waits for an SDP answer from the server. This SDP exchange should be able to include regular STUN/TURN candidates normally exchanged in WebRTC, but it does not need to include it.

2.3.2.2 Candidate Gathering

In the current version of the protocol, the client does not gather candidates prior to sending its SDP offer. Instead, it lets the server gather direct and relay candidates that will be sent back to the client in the SDP answer.

2.3.2.2.1 Relay Candidate Gathering

Relay candidate gathering begins with an initial list of relay candidates that can be obtained from the signaling server. This call to the signaling server creates the association id, and returns the information required to make JET accept requests.

The server endpoint **MUST** perform the accept requests on the original relay candidates to obtain the real list of relay candidates that will be sent to the client for connectivity testing. It is also important to properly parse the "Jet-Instance" response header of the JET accept request, since it corresponds to the correct relay server behind a potential load balancer.

2.3.2.2.2 Direct Candidate Gathering

Direct candidate gathering consists of obtaining the list of IP addresses and ports that the server is listening to on local network interfaces. Host candidate gathering is optional and can be disabled through a security policy since the list of IP addresses of a specific machine can be considered information disclosure.

The server must create the Association abstract data type and create candidate ids for each direct candidate. The result is a list of direct candidates that will be included in the SDP payload.