[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

Release: July 17th, 2019

Contents

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	1	2	3	4	5	6	7	8	9	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 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.

2.2.3.1 TCP Relay

The new TCP relay protocol can be assumed when the request path is not "/". The new protocol closely matches the WebSocket protocol, except without the WebSocket handshake and the usage of custom HTTP headers. The sequence always starts with the server connecting to the relay with an accept request. This accept request MUST include an association id that will be shared with the client through an external communication channel, such as a signaling server.

TCP relay servers MAY choose to accept any association id, and therefore requiring no authentication or synchronization with a table of association maintained by a signaling server. If authentication is desirable, it SHOULD be implemented by protecting the API endpoint used to create the original association and accepting only known association ids in the relay. The knowledge of the association id is considered sufficient to authenticate the accept and connect requests that follow, since it acts as a short-lived one-time token that was originally created through an authenticated API call.

2.2.3.1.1 TCP 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
```

2.2.3.1.2 TCP 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
```

2.2.3.2 TCP Direct

The TCP direct connectivity allows direct (non-relay) connectivity between a client and a server, offering better performance when such a route exists on the local network. TCP direct candidates are sent by the server to the client over an external communication channel such as a signaling server. Since the server endpoint does not need to connect to a relay, TCP direct candidates only require the client to attempt a connect request with the association id known to both peers.

A TCP direct server MUST reject unknown association ids to prevent selecting a candidate between the wrong client and server. This is particularly important since local network IP addresses are often the same between different networks. To prevent malicious a server from deliberately accepting requests not meant for them, the direct TCP connect response includes both the association and candidate ids. The client MUST validate that the candidate id is the same one that was previously shared through the external communication channel.

2.2.3.2.1 TCP 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
```

2.2.3.3 TLS Relay

A Jet TLS is similar to a TCP relay, with the exception that the TLS handshake is performed with the relay server and that the JET exchange occurs over TLS. In contrast, end-to-end TLS works over a TCP relay by relaying the traffic without decrypting the traffic. The TLS relay requires a certificate trusted by the client. Usage of a TLS relay implies that traffic inspection and recording is possible by the relay server, so trust goes beyond just a valid certificate.

There are two possible certificate configuration scenarios that can be envisioned for the TLS relay:

1) The certificate name matches the relay server, and validation expects the name to be the same as the relay server, just like a regular web server. In this case, a client connecting to "123456" through "jet.wayk.net" will expect "jet.wayk.net" as the certificate name.

2) The certificate name matches the target server, and validation expects the name to be the same as the target server. In this case, a client connecting to "123456" through "jet.wayk.net" will expect "123456" as the certificate name. This configuration type required a custom root CA that can emit new certificates for any name, using server-name indication (SNI) to figure out what the name should be.

The first configuration scenario is the recommended one, since it is the easiest one to implement and it can work without a custom root CA. In both cases, server-name indication MUST match the intended name for validation.

2.2.3.4 TLS Direct

A Jet TLS direct candidate is very similar to a Jet TCP direct candidate, which the exception that the JET exchange is encrypted. Unlike the TLS relay candidate, the TLS direct candidate works with end-to-end TLS. Server-name indication SHOULD be used to select between a certificate matching the machine name on the local network, and a certificate emitted by a custom root CA trusted by both peers with a name matching the one used by the peer in a signaling server.

2.2.3.5 TCP Relay (Old)

The original TCP relay protocol used "/" as the request path and encoded the method inside the "Jet-Method" HTTP header. The TCP accept method did not specify an association id in the request, and instead created an association with the id returned in the accept response. For compatibility reasons, the “/” request path can be used to detect an older implementation.

2.2.3.5.1 TCP Accept

>> Request:

```
GET / HTTP/1.1
Host: jet.wayk.net
Connection: Keep-Alive
Jet-Method: Accept
Jet-Version: 1
```

<< Response:

```
HTTP/1.1 200 OK
Jet-Association: e6ec698c-5793-4c63-af79-bd644ccf022f
Jet-Instance: jet101.wayk.net:8080
Jet-Version: 1
```

2.2.3.5.2 TCP Connect

>> Request:

```
GET / HTTP/1.1
Host: jet101.wayk.net
Connection: Keep-Alive
Jet-Method: Connect
Jet-Association: e6ec698c-5793-4c63-af79-bd644ccf022f
Jet-Version: 1
```

<< Response:

```
HTTP/1.1 200 OK
Jet-Version: 1
```

2.2.4 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.

2.2.4.1 WebSocket Server 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=
```

2.2.4.2 WebSocket Client 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=

2.3 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.