



monotch

SMART MOBILITY PLATFORMS

TLEX-BROKER-SYSTEM Interface v1.1.1

Table of contents

| | | |
|-------|----------------------------------------|----|
| 1 | Versioning..... | 3 |
| 2 | Referenced documents | 4 |
| 3 | Introduction..... | 5 |
| 4 | Overview | 6 |
| 5 | API..... | 7 |
| 5.1 | Authentication and authorization | 7 |
| 5.2 | API endpoints | 7 |
| 5.2.1 | Sessions..... | 8 |
| 5.2.2 | TLCs..... | 20 |
| 6 | TCPStreaming protocol | 24 |
| 6.1 | Transport Layer Security | 24 |
| 6.2 | Protocol version establishment | 24 |
| 6.3 | Protocol version 0x01..... | 25 |
| 6.3.1 | Byte Order | 25 |
| 6.3.2 | Datagram framing..... | 25 |
| 6.3.3 | Communication modes..... | 26 |
| 6.3.4 | Datagrams..... | 30 |

1 Versioning

This document is using a versioning scheme that indicates the version of this TLEX interface and tracks the revisions of this document. This version scheme is <interface version major>.<interface version minor>.<document revision>. The first two version numbers (major and minor) indicate the version of the interface and only change when there is technical change in the described interface. Major version is only bumped when there is compatibility breaking change. Minor version is bumped on trivial, non breaking changes of the interface. The last version number indicates the revision of this document.

| Version | Date | Author | Changes |
|---------|-------------|--------------|-------------------------------------------------------------------------------|
| 1.0.0 | 13 Mar 2017 | L. Rijneveld | Initial specification for TLEX release v1.0 |
| 1.1.0 | 25 Jul 2017 | L. Rijneveld | Update for TLEX release v1.1 TLC type field in response of /tlcs end point |
| 1.1.1 | 23 Mar 2020 | L. Rijneveld | Improved layout and formatting |

2 Referenced documents

| ID | Reference | Version | Date |
|-----|-----------------------------|---------|-------------|
| [1] | TLEX-BROKER-ADMIN Interface | 1.1.1 | 23 Mar 2020 |

3 Introduction

This document describes the technical interface with TLEX for Broker systems . The primary goal of this document is to track/version the interface specification for this explicit context so that the impact of future TLEX specification changes can be properly assessed.

4 Overview

Broker systems interface with TLEX on two levels:

1. JSON-REST API; used for:
 - a. requesting registered TLCs;
 - b. requesting/creating TCPStreaming sessions;
 - c. updating TCPStreaming sessions.
2. TCPStreaming protocol: TCP protocol used for sending and receiving payloads to and from TLEX.



5 API

5.1 Authentication and authorization

The authentication of the Client will be based on a "authorization token". This "authorization token" needs to be passed as the "X-Authorization" request header value. The authorization token needs to belong to an "BROKER_SYSTEM" authorization (for more details the "authorization model" chapter in the TLEX-BROKER-ADMIN Interface document [1]).

5.2 API endpoints

| End point | Method | URI | Description |
|-----------|--------|-------------------|-----------------------------------------|
| sessions | POST | /sessions | Creates a new streaming session |
| sessions | GET | /sessions | Retrieves all active streaming sessions |
| sessions | GET | /sessions/<token> | Retrieves one active streaming session |
| sessions | PUT | /sessions/<token> | Updates one active streaming session |
| tlcs | GET | /tlcs | Gets all TLC registrations |
| tlcs | GET | /tlcs/<uuid> | Retrieve one specific TLC registration |

5.2.1 Sessions

5.2.1.1 POST /sessions

Creates a new streaming session.

5.2.1.1.1 Request

```
POST <API base URL>/sessions HTTP/1.1
Host: <hostname>
X-Authorization: <authorization token>
Content-Type: application/json
```


```
{
  "domain": "<domain>",
  "type": "<type>",
  "protocol": "<protocol>",
  "details": {
    <protocol details>
  }
}
```

| Name | Description |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| domain | Sessions are created within a specific domain, identified by a single string Only sessions created for the same domain will be able to stream data to each other |
| type | The session type; must be "Broker" |
| protocol | The session protocol; must be "TCPStreaming_Multiplex" |
| details | Session protocol specific details for creating the session |

5.2.1.1.2 Response

HTTP/1.1 200 OK
 Content-Type: application/json

```
{
  "token": "<token>",
  "domain": "<domain>",
  "type": "<type>",
  "protocol": "<protocol>",
  "details": {
    <protocol details>
  }
}
```

| Name | Description |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| token | The token for the created session <div>  This token is unique and can only be used once for establishing a TCP connection; if the session expires or ends (TCP disconnect) a new session needs be created to obtain a new token </div> |
| domain | See request |
| type | See request |
| protocol | See request |
| details | Session protocol specific details of the created session |

5.2.1.1.3 Session type "Broker" with protocol "TCPStreaming_Multiplex"

TCP based multiplex streaming session for a payload broker.

Payloads sent by the client will be received by "TLC" session clients if the payload "TLC identifier" is within their scope.

Payloads sent by "TLC" session clients having a payload "TLC identifier" that is within the scope of the session will be received.

5.2.1.1.3.1 Request details

```
{  
  "securityMode": "<security mode>",  
  "tlcIdentifiers": [<"TLC identifier">, "<TLC identifier">, ...]  
}
```

| Name | Description |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| securityMode | Security mode of the streaming session Must be one of the predefined values: 1. NONE 2. TLSv1.2 |
| tlcIdentifiers | The TLC identifiers for the session Since the session is for multiple TLC's, payload data will be streamed with TLC identifier (see protocol datagram 0x05) |

5.2.1.1.3.2 Response details

```
{
  "securityMode": "<security mode>",
  "tlcIdentifiers": ["<TLC identifier>", "<TLC identifier>", ...]
  "listener": {
    "host": "<host address>",
    "port": <port number>,
    "expiration": "<ISO 8601 date time>"
  },
  "keepAliveTimeout": "<ISO 8601 duration>",
  "clockDiffLimit": "<ISO 8601 duration>",
  "clockDiffLimitDuration": "<ISO 8601 duration>",
  "payloadRateLimit": <payload/second limit>,
  "payloadRateLimitDuration": "<ISO 8601 duration>",
  "payloadThroughputLimit": <KB/second limit>,
  "payloadThroughputLimitDuration": "<ISO 8601 duration>"
}
```

| Name | Description |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| securityMode | See request details |
| tlcIdentifiers | See request details |
| listener | The Streaming Service Node listener details for establishing the TCP connection |
| listener.host | The host address of the Streaming Service Node |
| listener.port | The TCP port of the Streaming Service Node's session listener |
| listener.expiration | <p>The expiration date time of the listener in ISO 8601 date time format</p> <p>If the TCP connection has not been established before this time the listener will expire and the streaming session will no longer be available</p> <p>The default listener expiration will be 5 seconds after creation</p> |
| keepAliveTimeout | <p>The keep alive timeout duration of the TCP connection in ISO 8601 duration format</p> <p>If no TCP data is received during the specified duration, the TCP connection will be terminated by the Streaming Service</p> <p>If a Client receives no TCP data during the specified duration, it must terminate the TCP connection</p> |

| Name | Description |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| clockDiffLimit | <p>The maximum clock difference, in ISO 8601 duration format, allowed for the streaming session</p> <p>If the average clock difference during the duration (see clockDiffLimitDuration) exceeds the limit the Streaming Service will terminate the TCP connection</p> |
| clockDiffLimitDuration | <p>The period, in ISO 8601 duration format, during which the average clock difference should not exceed the clockDiffLimit</p> |
| payloadRateLimit | <p>The maximum amount of payloads per second allowed for the streaming session</p> <p>If the average amount of received payloads per second during the duration (see payloadRateLimitDuration) exceeds the limit the Streaming Service will terminate the TCP connection</p> |
| payloadRateLimitDuration | <p>The period, in ISO 8601 duration format, during which the average amount of received payloads per second should not exceed the payloadRateLimit</p> |
| payloadThroughputLimit | <p>The maximum amount of payload kilobytes (KB) per second allowed for the streaming session</p> <p>If the average amount of received payload kilobytes (KB) per second during the duration (see payloadThroughputLimitDuration) exceeds the limit the Streaming Service will terminate the TCP connection</p> |
| payloadThroughputLimitDuration | <p>The period, in ISO 8601 duration format, during which the average amount of received payload kilobytes (KB) per second should not exceed the payloadThroughputLimit</p> |

5.2.1.1.3.3 Example

```
POST api/v1/sessions HTTP/1.1
Host: api.tlex.eu
X-Authorization: dtNB_vhvJ0wgTGf1N0DxN38_AmTL_4yiPRZdqZSuK3k
Content-Type: application/json

{
  "domain": "test",
  "type": "Broker",
  "protocol": "TCPStreaming_Multiplex",
  "details": {
    "securityMode": "NONE",
    "tlcIdentifiers": ["NLZH0023", "NLZH0024", "NLZH0025"]
  }
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "token": "cXXrqTkreh0vLbuuYKKQQGAU1MTGGGBC1N1izwYaqu8",
  "domain": "test",
  "type": "Broker",
  "protocol": "TCPStreaming_Multiplex",
  "details": {
    "securityMode": "NONE",
    "tlcIdentifiers": ["NLZH0023", "NLZH0024", "NLZH0025"],
    "listener": {
      "host": "n11.tlex.eu",
      "port": 40344,
      "expiration": "2016-11-17T16:07:56Z"
    },
    "keepAliveTimeout": "PT5S",
    "clockDiffLimit": "PT3S",
    "clockDiffLimitDuration": "PT60S",
    "payloadRateLimit": 1200,
    "payloadRateLimitDuration": "PT5S",
    "payloadThroughputLimit": 120,
    "payloadThroughputLimitDuration": "PT5S"
  }
}
```

5.2.1.2 GET /sessions

Retrieves all active streaming sessions.

Intended for monitoring and debug purposes.

5.2.1.2.1 Request

```
GET <API base URL>/sessions HTTP/1.1
Host: <hostname>
X-Authorization: <authorization token>
Content-Type: application/json
```

5.2.1.2.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  <session>, <session>
]
```

5.2.1.2.3 Example

```
GET api/v1/sessions HTTP/1.1
Host: api.tlex.eu
X-Authorization: dtNB_vhvJ0wgTGf1N0DxN38_AmTL_4yiPRZdqZSuK3k
Content-Type: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
[
  {
    "token": "cXXrqTkreh0vLbuuYKKQGAU1MTGGGBC1N1izwYaqu8",
    "domain": "test",
    "type": "Broker",
    "protocol": "TCPStreaming_Multiplex",
    "details": {
      "securityMode": "NONE",
      "tlcIdentifiers": ["NLZH0023", "NLZH0024", "NLZH0025"],
      "listener": {
        "host": "n44.tlex.eu",
        "port": 40344,
        "expiration": "2016-11-17T16:07:56Z"
      },
      "keepAliveTimeout": "PT5S",
      "clockDiffLimit": "PT3S",
      "clockDiffLimitDuration": "PT60S",
      "payloadRateLimit": 1200,
      "payloadRateLimitDuration": "PT5S",
      "payloadThroughputLimit": 120,
      "payloadThroughputLimitDuration": "PT5S"
    }
  }
]
```

5.2.1.3 GET /sessions/<token>

Retrieves the active streaming session with the given "token".
Intended for monitoring and debug purposes.

5.2.1.3.1 Request

```
GET <API base URL>/sessions/<session token> HTTP/1.1
Host: <hostname>
X-Authorization: <authorization token>
Content-Type: application/json
```

5.2.1.3.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/json

<session>
```


5.2.1.3.3 Example

```
GET api/v1/sessions/cXXrqTkrehOvLbuuYKKQQGAU1MTGGGBC1N1izwYaqu8 HTTP/1.1
Host: api.tlex.eu
X-Authorization: dtNB_vhvJ0wgTGf1N0DxN38_AmTL_4yiPRZdqZSuK3k
Content-Type: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "token": "cXXrqTkrehOvLbuuYKKQQGAU1MTGGGBC1N1izwYaqu8",
  "domain": "test",
  "type": "Broker",
  "protocol": "TCPStreaming_Multiplex",
  "details": {
    "securityMode": "NONE",
    "tlcIdentifiers": ["NLZH0023", "NLZH0024", "NLZH0025"],
    "listener": {
      "host": "n44.tlex.eu",
      "port": 40344,
      "expiration": "2016-11-17T16:07:56Z"
    },
    "keepAliveTimeout": "PT5S",
    "clockDiffLimit": "PT3S",
    "clockDiffLimitDuration": "PT60S",
    "payloadRateLimit": 1200,
    "payloadRateLimitDuration": "PT5S",
    "payloadThroughputLimit": 120,
    "payloadThroughputLimitDuration": "PT5S"
  }
}
```

5.2.1.4 PUT /sessions/<token>

Updates the protocol details of the active streaming session with the given "token".
Intended to support the addition and removal of TLC identifiers for multiplex sessions.

5.2.1.4.1 Request

```
PUT <API base URL>/sessions/<session token> HTTP/1.1
Host: <hostname>
X-Authorization: <authorization token>
Content-Type: application/json
```

5.2.1.4.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/json

<protocol details>
```

5.2.1.4.3 Example

```
PUT api/v1/sessions/cXXrqTkrehOvLbuuYKKQQGAU1MTGGGBC1N1izwYaqu8 HTTP/1.1
Host: api.tlex.eu
X-Authorization: dtNB_vhvJ0wgTGf1N0DxN38_AmTL_4yiPRZdqZSuK3k
Content-Type: application/json
```

```
{
  "securityMode": "NONE",
  "tlcIdentifiers": ["NLZH0023", "NLZH0026"]
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "token": "cXXrqTkrehOvLbuuYKKQQGAU1MTGGGBC1N1izwYaqu8",
  "domain": "test",
  "type": "Broker",
  "protocol": "TCPStreaming_Multiplex",
  "details": {
    "securityMode": "NONE",
    "tlcIdentifiers": ["NLZH0023", "NLZH0026"],
    "listener": {
      "host": "n44.tlex.eu",
      "port": 40344,
      "expiration": "2016-11-17T16:07:56Z"
    },
    "keepAliveTimeout": "PT5S",
    "clockDiffLimit": "PT3S",
    "clockDiffLimitDuration": "PT60S",
    "payloadRateLimit": 1200,
    "payloadRateLimitDuration": "PT5S",
    "payloadThroughputLimit": 120,
    "payloadThroughputLimitDuration": "PT5S"
  }
}
```

5.2.2 TLCs

5.2.2.1 GET /tlcs

Retreives all TLC registrations.

Intended for supporting dynamic setup of multiple load balancing "TCPStreaming_Multiplex" sessions without having to maintain a static "TLC identifier" list.

5.2.2.1.1 Request

```
GET <API base URL>/tlcs HTTP/1.1
Host: <hostname>
X-Authorization: <authorization token>
Content-Type: application/json
```

5.2.2.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  <TLC>, <TLC>, ...
]
{
  "uuid": "<TLC uuid>",
  "identifier": "<TLC identifier>",
  "type": "<TLC type>",
  "domain": "<domain>",
  "account": "<account>"
}
```

| Name | Description |
|------------|--------------------------------------------------------------------------------------------------------------------------------------|
| uuid | The unique id for the created TLC |
| identifier | The TLC identifier of the TLC |
| type | Type of TLC; must be one of the predefined types: <ol style="list-style-type: none">1. TCPStreaming2. VLOG |

| Name | Description |
|---------|-----------------------------------------------------------|
| domain | The domain in which the TLC is registered |
| account | Unique id of the account that "owns" the TLC registration |

5.2.2.1.3 Example

```
GET api/v1/tlcs HTTP/1.1
Host: api.tlex.eu
X-Authorization: dtNB_vhvJ0wgTGf1N0DxN38_AmTL_4yiPRZdqZSuK3k
Content-Type: application/json
HTTP/1.1 200 OK
Content-Type: application/json
```

```
[
  {
    "uuid": "4aa1ace8-32b0-42b6-925a-7d7a33e97859",
    "identifier": "tlc_0001",
    "type": "TCPStreaming",
    "domain": "test",
    "account": "80b142ab-88e8-4600-9a86-8807c19b1b2a"
  },
  {
    "uuid": "d1c9ca3d-23e1-4191-bfa3-b8364c52a4ce",
    "identifier": "tlc_0002",
    "type": "TCPStreaming",
    "domain": "test",
    "account": "80b142ab-88e8-4600-9a86-8807c19b1b2a"
  }
]
```

5.2.2.2 GET /tlcs/<uuid>

Retreives the TLC registration with the given "uuid".

5.2.2.2.1 Request

```
GET <API base URL>/tlcs/<TLC uuid> HTTP/1.1
Host: <hostname>
X-Authorization: <authorization token>
Content-Type: application/json
```

5.2.2.2.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

<TLC>

5.2.2.2.3 Example

```
GET api/v1/tlcs/4aa1ace8-32b0-42b6-925a-7d7a33e97859 HTTP/1.1
Host: api.tlex.eu
X-Authorization: dtNB_vhvJ0wgTGf1N0DxN38_AmTL_4yiPRZdqZSuK3k
Content-Type: application/json
HTTP/1.1 200 OK
Content-Type: application/json

{
  "uuid": "4aa1ace8-32b0-42b6-925a-7d7a33e97859",
  "identifier": "tlc_0001",
  "type": "TCPStreaming",
  "domain": "test",
  "account": "80b142ab-88e8-4600-9a86-8807c19b1b2a"
}
```

6 TCPStreaming protocol

The TCPStreaming protocol facilitates a continuous asynchronous bi-directional stream of datagrams.

6.1 Transport Layer Security

The Streaming Service optionally uses Transport Layer Security version 1.2 (TLSv1.2) to secure the TCP communication used for TCP streaming protocol sessions. The only supported cipher suite is "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256". Only server side authentication will be used.

6.2 Protocol version establishment

To support future protocol enhancement or replacement a one-byte version identifier is sent before the datagram streaming starts. When the peer does not support the protocol version the connection should be closed by the peer.

| Protocol version (1 byte) |
|---------------------------|
| 0x01 |

6.3 Protocol version 0x01

In protocol version 0x01 datagrams are streamed by using frames with a 4 byte header which exists out of a 2 byte fixed prefix and a 2 byte frame data size. The header is followed by the frame data containing the actual datagram.


6.3.1 Byte Order

The byte order used by the protocol is "big-endian" also known as "network byte order": the byte containing the most significant bit (MSB) will be transmitted first.

6.3.2 Datagram framing

The datagrams are framed by having a 2 byte data size indication in the header. The header prefix, the fixed bytes 0xAA and 0xBB, has been added for "out of sync" detection and also to have 32 bit "aligned" header. By having the data size in the header no stream decoding is needed to read subsequent frames. Since the header data size is 2 bytes, one frame can hold a maximum of 63KB of data. The size of the frames should always be > 0; empty frames are not supported.

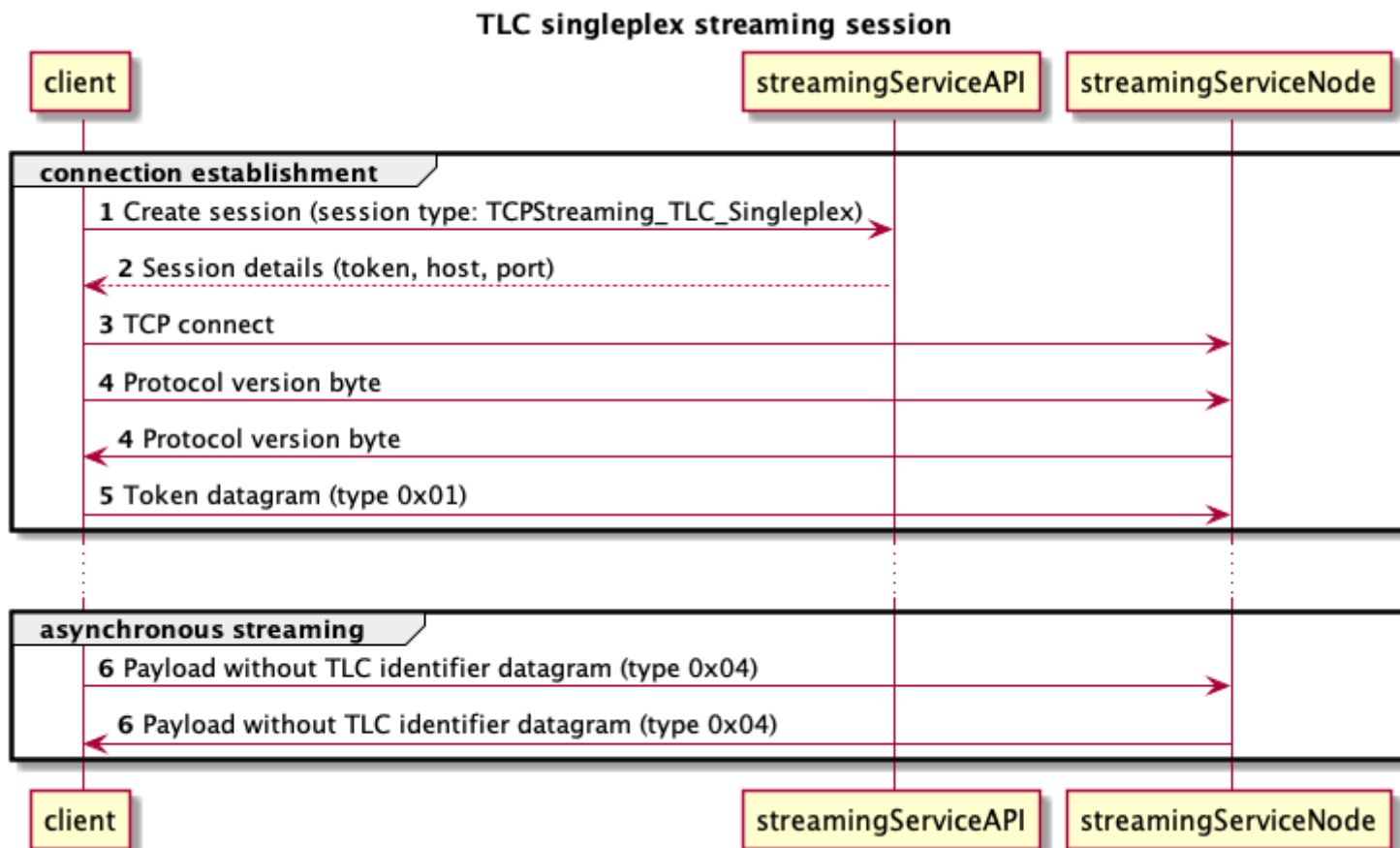
| Version preamble (1 byte) | Frame 1 | | | Frame 2 | | | Frame n | | |
|---------------------------|---------|-----------|------|---------|-----------|---------------|---------|-----------|------|
| | Header | | Data | Header | | Data | Header | | Data |
| | Prefix | Data size | | Prefix | Data size | | Prefix | Data size | |
| 0x01 | 0xAA BB | 0x00 01 | 0x00 | 0xAA BB | 0x00 04 | 0x04 01 03 23 | 0xAA BB | ... | ... |

 The fixed header prefix bytes should always be checked (asserted) since a mismatch would indicate a framing issue resulting in data corruption. When such a framing issue is detected the connection should be terminated as soon as possible.

6.3.3 Communication modes

6.3.3.1 TLC singleplex mode

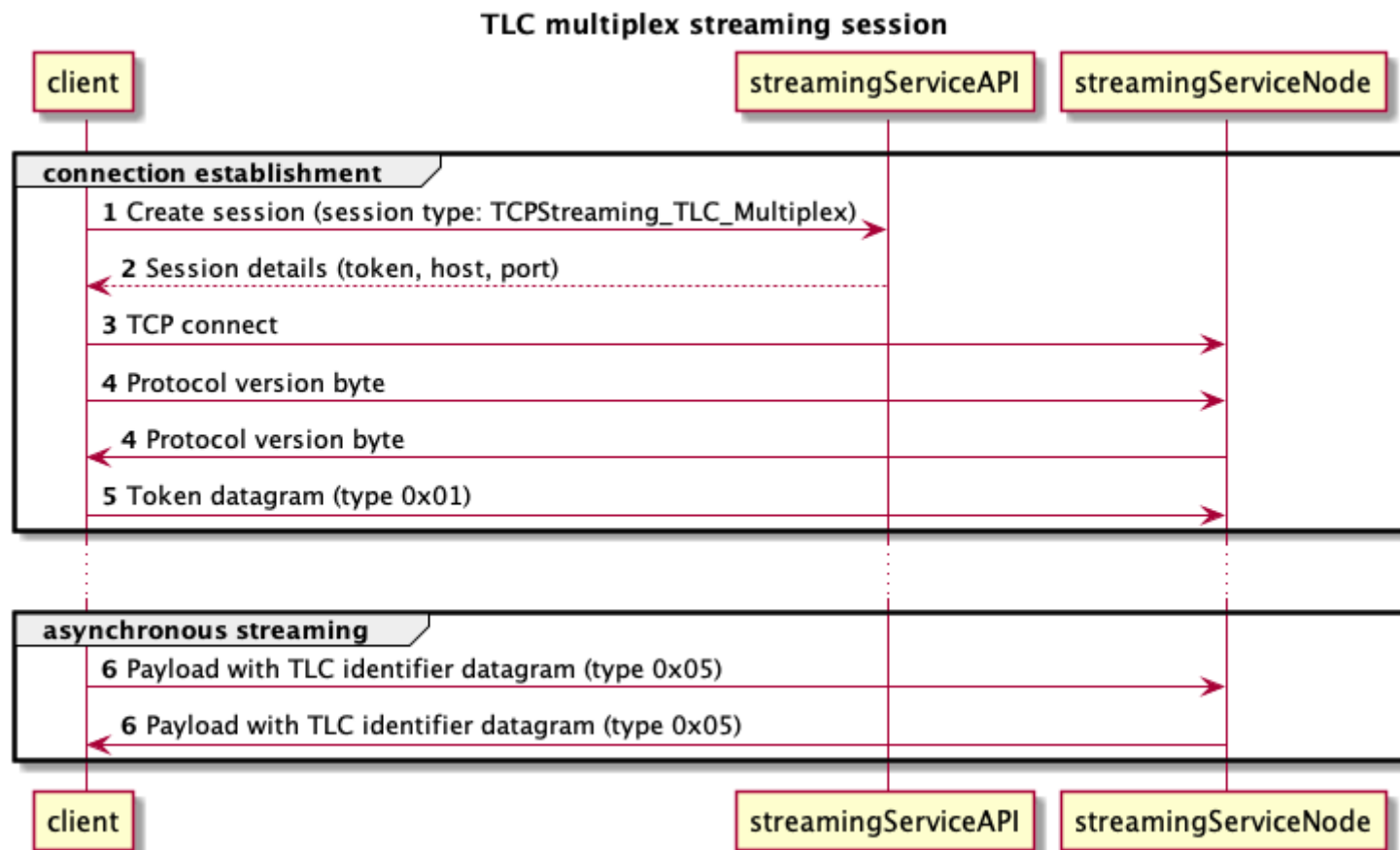
This mode can be used by TLC systems. Streaming sessions of this type will receive all payloads from "Broker" sessions when the "TLC identifier" matches the "TLC identifier" specified in the "sessions" API call used for creating the streaming session. Only one session per "TLC identifier" will be allowed.



6.3.3.2 TLC multiplex mode

This mode can be used by TLC systems. Streaming session in this mode will receive all payloads from "Broker" sessions when the "TLC identifier" matches one of the "TLC identifiers" specified in the "sessions" API call used for creating the streaming session.

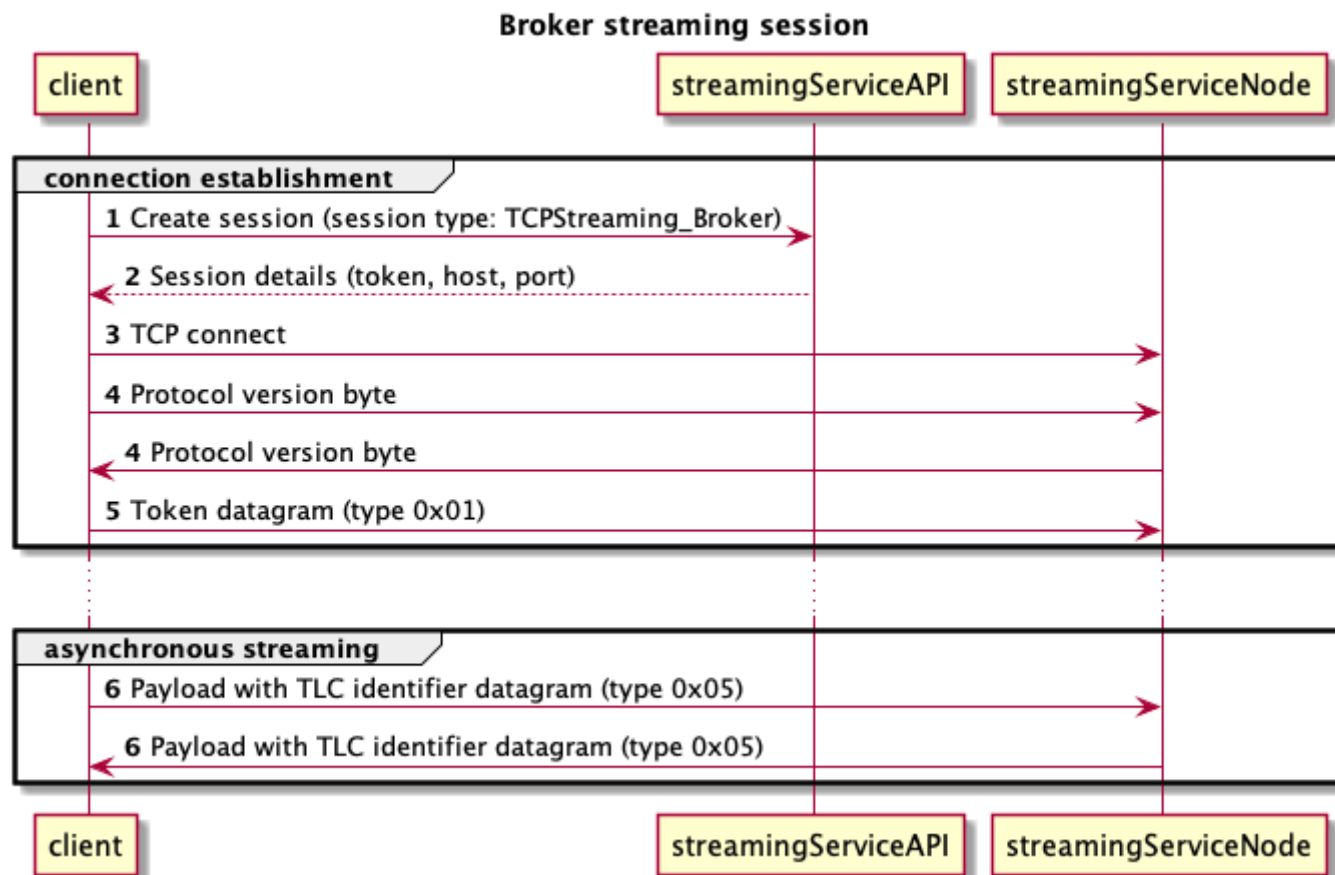
⚠ If one of the "TLC identifiers" in the "create session" call is already being used in an other active session, the API will refuse the "create session" call.



6.3.3.3 Broker mode

This mode will be used by the Broker systems. Streaming session in this mode will receive all payloads from "TLC singleplex" or "TLC multiplex" sessions when the "TLC identifier" matches one of the "TLC identifiers" specified in the "sessions" API call used for creating the streaming session.

⚠ If one of the "TLC identifiers" in the "create session" call is already being used in an other active session of the Broker account, the API will refuse the "create session" call.

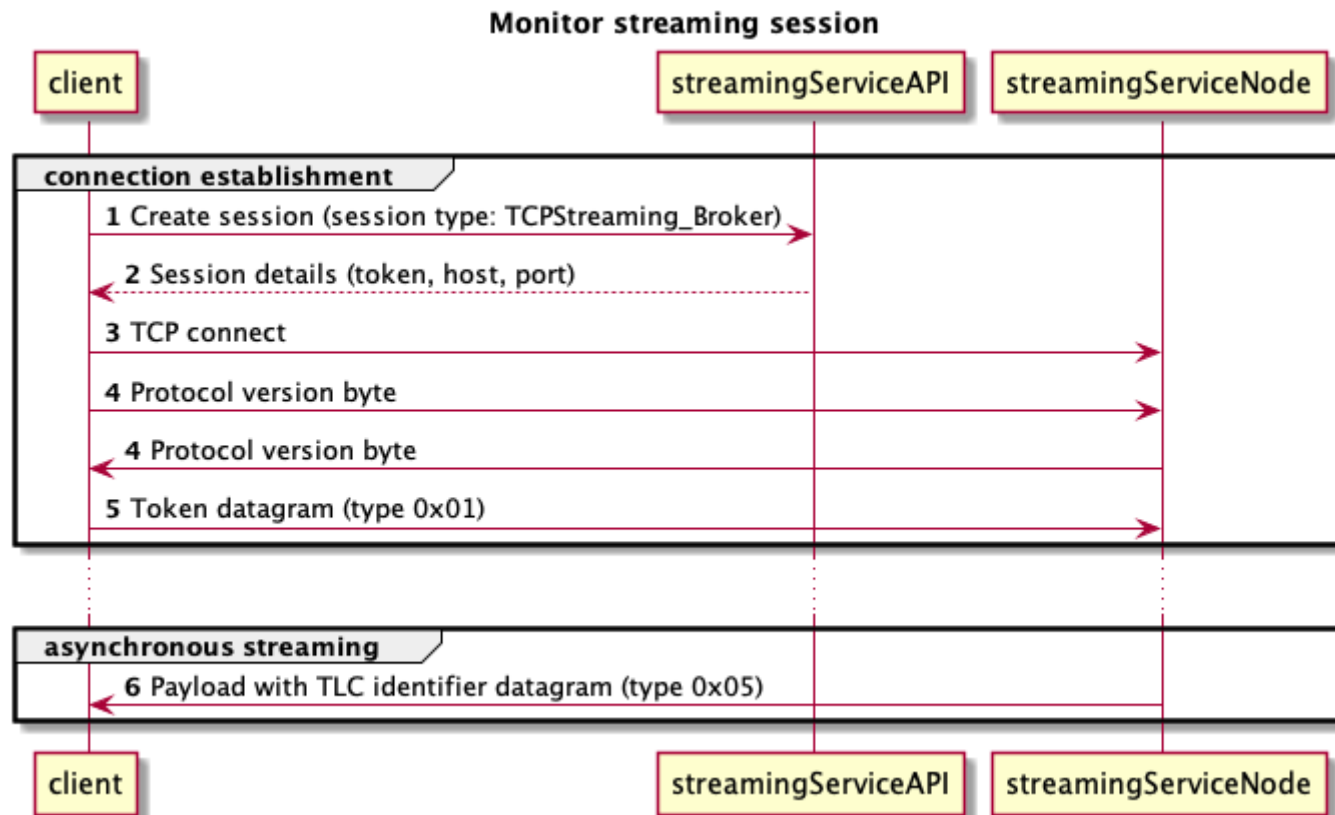


6.3.3.4 Monitor mode

This mode will be used by governance organisations to facilitate analytics and diagnostics. Streaming session in this mode will receive all payloads from "TLC singleplex", "TLC multiplex" sessions and "Broker" sessions when the "TLC identifier" matches one of the "TLC identifiers" specified in the "sessions" API call used for creating the streaming session.

⚠ If one of the "TLC identifiers" in the "create session" call is already being used in an other active session of the organisation (account), the API will refuse the "create session" call.

⚠ Monitor sessions can only receive data.



6.3.4 Datagrams

The frame data contains the actual datagram which is identified by a one byte "datagram type" identifier. The following datagrams are defined:

6.3.4.1 KeepAlive (0x00)

This datagram can be used to prevent disconnects when no data is available to sent within the keep alive timeout.

| Datagram type (1 byte) |
|------------------------|
| 0x00 |

6.3.4.2 Token (0x01)

The first datagram sent by the Client should be "Token" in order the properly authenticate the Client.

| Datagram type (1 byte) | Token (ASCII encoded) |
|------------------------|-----------------------|
| 0x01 | <Token> |


6.3.4.3 Bye (0x02)

This datagram can be used by either party to indicate that the connection will be closed. The reason (ASCII Encoded) is optional. It is the last datagram sent.

| Datagram type (1 byte) | Disconnect reason (optional, ASCII encoded) |
|------------------------|---------------------------------------------|
| 0x02 | <Reason> |

6.3.4.4 Reconnect (0x03)

This datagram can be used by the Streaming Service to instruct the Client to reconnect as soon as possible.

 A new session needs to be created using the API in order to reconnect.

| Datagram type (1 byte) |
|------------------------|
| 0x03 |

6.3.4.5 Payload datagrams (0x04, 0x05)

The payload datagrams "carry" the actual payload which is being streamed by the Streaming Service. Both payload datagrams have an "Origin timestamp" field that should be filled with transmission time timestamp. This timestamp must be obtained from the same clock source as used to handle the "timestamps request" message. The payload type is identified by a single payload type byte. Payload type bytes "0xF0" to "0xFF" are reserved for protocol use.


6.3.4.5.1 Protocol payload types

| Payload type byte | Payload type |
|-------------------|------------------------------------------------------|
| 0xF0 | Monitor Payload (see Monitor Payload Encoding below) |

6.3.4.5.2 Payload without TLC identifier (0x04)

This datagram is in "TLC_Singleplex" communication mode to send and receive data and can only be used for singleplex sessions (session type "TCPStreaming_TLC_Singleplex").


| Datagram type (1 byte) | Payload type byte (1 byte) | Origin timestamp (8 bytes) | Payload (n bytes) |
|------------------------|----------------------------|----------------------------|-------------------|
| 0x04 | <Payload type byte> | <UTC milliseconds> | <Payload> |

 If this datagram is used for a multiplex session, the session will be terminated by the Streaming Service.

6.3.4.5.3 Payload with TLC identifier (0x05)

This datagram is used in "TLC_Multiplex" and "Broker" communication modes to send and receive data in a multiplex session (session types "TCPStreaming_TLC_Multiplex" and "TCPStreaming_Broker").

| Datagram type (1 byte) | TLC identifier (8 bytes, ASCII encoded) | Payload type byte (1 byte) | Origin timestamp (8 bytes) | Payload (n bytes) |
|------------------------|-----------------------------------------|----------------------------|----------------------------|-------------------|
| 0x05 | <TLC identifier> | <Payload type byte> | <UTC milliseconds> | <Payload> |

 In communication modes "TLC_Multiplex" and "Broker" only datagrams with a "TLC identifier" requested for the streaming session are allowed. If a non-requested "TLC identifier" is used, the payload will be dropped by the Streaming Service.

6.3.4.5.4 Monitor Payload Encoding

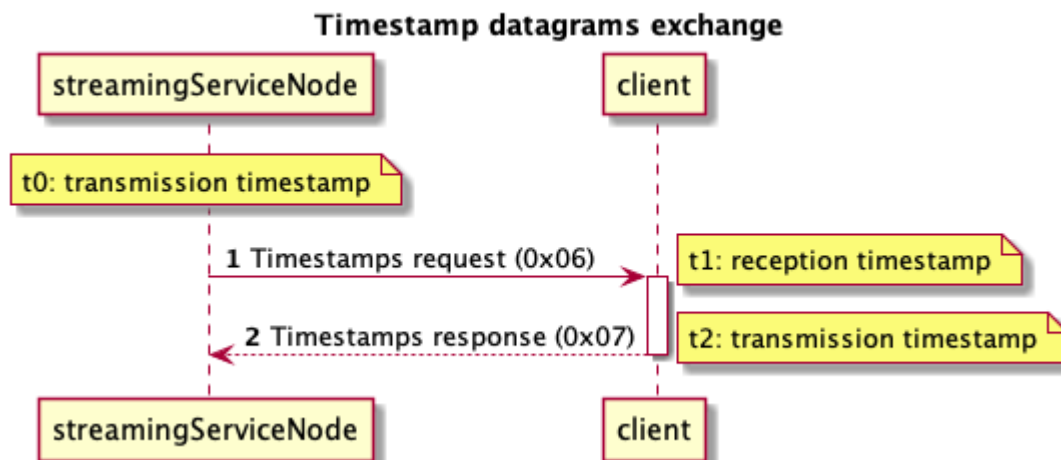
Payload type 0xF0 indicates a special payload that adds a diagnostic header to the original payload with additional diagnostic information.

| Publisher session token length (4 byte) | Publisher session token (n bytes, ASCII encoded) | Publishing timestamp (8 bytes) | Sent timestamp (8 bytes) | Original payload type (1 byte) | Original payload (n bytes) |
|-----------------------------------------|--------------------------------------------------|--------------------------------|--------------------------|--------------------------------|----------------------------|
| <length in bytes> | <Publisher session token> | <UTC milliseconds> | <UTC milliseconds> | <Payload type> | <Payload> |

 It is possible for the Publisher session token to have a "zero length". This indicates a payload that has been resent instead of being published by a publisher.

6.3.4.6 Timestamp datagrams (0x06, 0x07)

The timestamp datagrams are used to determine connection latency and clock difference.



6.3.4.6.1 Timestamps request (0x06)

This datagram contains the transmission timestamp which will be "echo'd back" by the recipient using the "Timestamps response" datagram. The Streaming Service will send this datagram at a regular interval (15 seconds) to measure the connection latency and to calculate the clock difference between the Streaming Service and the connected party. The connection latency is used to evaluate the accuracy of the calculated clock difference.

| Datagram type (1 byte) | t0: transmission timestamp (8 bytes) |
|------------------------|--------------------------------------|
| 0x06 | <UTC milliseconds> |

6.3.4.6.2 Timestamps response (0x07)

This datagram must be used as response to a received "Timestamps request" datagram and must contain the original "t0: transmission timestamp" of this datagram. The response should be prioritised to any other traffic; in other words: it should be sent as soon as possible after receiving the "Timestamps request" message. The "t1: request reception timestamp" field must be filled with the reception time timestamp and should be determined as soon as possible upon reception of the "Timestamps request" message. The "t2: response transmission timestamp" field must be filled with the transmission time timestamp and should be determined as late as possible in the transmission process.

| Datagram type (1 byte) | t0: transmission timestamp from request (8 bytes) | t1: request reception timestamp (8 bytes) | t2: response transmission timestamp (8 bytes) |
|------------------------|---------------------------------------------------|-------------------------------------------|-----------------------------------------------|
| 0x07 | <UTC milliseconds> | <UTC milliseconds> | <UTC milliseconds> |