Network Protocol Specification

Version 0.9

Scenera, Inc., Nikon Corporation, Sony Semiconductor Solutions Corporation, Wistron Corporation and Hon Hai Precision Industry Co., Ltd.(NICE Alliance Promoters) contributed to this document.

Revision History

Version	Date	Comments
0.9rc1	13 Nov 2018	First draft
0.9rc2	25 Feb 2019	Second draft
0.9	25 Mar 2019	Final draft

Contributors

Name	**Company**
Andrew Wajs	Scenera
Aviram Cohen	Scenera
Munehiro Shimomura	Sony
Hironori Miyoshi	Sony
Wendy Tin	Wistron

Table of Contents

1. Scope

This document provides the network protocols utilized commonly in the NICE System except backend process. It also describes the message structure to transfer data between Entities in NICE.

2. Overview

The Entity in NICE System have the three kinds of Interfaces - **Management**, **Control** and **Data**. This documents defines the followings to use the Interface.

- Network protocols
- Mechanism to identify the Entities
- Message format to exchange data between Entities over the Interface.

The following show the network protocols that Entity must support.

	HTTPS	MQTT	WebRTC
Management	X	X	
Control	X	X	X
Data	X	X	X

As for Control and Data interface, the following combination is allowed.

Interface		Availability	Application (Informative)
Control	**Data**		
WebAPI	WebAPI	X	Enterprise use (local network only)
	MQTT	X	Enterprise use
	WebRTC	N/A*	
MQTT	WebAPI	X	Typical use
	MQTT	X	
	WebRTC	N/A*	
WebRTC	WebAPI	N/A*	
	MQTT	N/A*	
	WebRTC	X	App-to-Device direct (Serverless)

N/A* : This combination is not applicable because DataChannel is used in both Control and Data in WebRTC.

3. Destination ID

Destination ID is to identify the destination Entity in Management, Control and Data Interface. The format is compliant with Uniform Resource Identifier (URI) RFC 3986. Naming convention is as follows:

Format: **[Scheme][Authority][Path]**

- [Scheme]
  - Network protocol
    - "mqtt://" in case of MQTT.
    - "webrtc://" in case of WebRTC.
    - "https://" in case of WebAPI.
    - "local://" is allowed to use the proprietary protocol as long as ensure the transport security in case that the node connection is in an Entity.
- [Authority]
  - MQTT Broker address is presented in case of MQTT.
  - HTTP Server address is presented in case of WebAPI.
  - Not applicable in case of WebRTC.
- [Path]
  - Series of the following elements.
    - [VERSION]
      - NICE Specification version.
    - [EndPointID]
      - Unique identifier of the destination Entity, which is assigned by NICE System. Format is defined in NICE Identifier Structure specification.
    - [NODE_ID]
      - Unique identifier of the Node in the Entity, which is assigned by the Entity itself. Format is defined in NICE Identifier Structure specification.
      - "0000" is used in case that NodeID is not applicable.
    - [API_NAME]
      - API name which Entity supports.
    - [PORT_ID]
      - Identifier to distinguish the stream data. It is equivalent to PortID of Node. Format is defined in NICE Identifier Structure specification.

Interface	Name convention of [Path]
Management	/[VERSION]/[EndPointID]/**management**/[API_NAME]
Control	/[VERSION]/[EndPointID]/**control**/[NODE_ID]/[API_NAME]
Data	/[VERSION]/[EndPointID]/**data**/[NODE_ID]/[PORT_ID]

Destination ID is applied to each protocol as following.

Protocol		Interface	Value	Example
MQTT	Broker and Topic	Management	mqtt://[Authority]/[VERSION]/[EndPointID]/management/[API_NAME]	mqtt://broker1.nicealliance.com/1/uuid/management/WebRTCSignaling
		Control	mqtt://[Authority]/[VERSION]/[EndPointID]/control/[NODE_ID]/[API_NAME]	mqtt://broker2.nicealliance.com/1/uuid/control/0000/StartStreaming

		Data	mqtt://[Authority]/[VERSION]/[EndPointID]/data/[NODE_ID]/[PORT_ID]	mqtt://broker3.nicealliance.com/1/uuid/data/0001/0000
WebRTC	Label of Data Channel	Control	webrtc:///[VERSION]/[EndPointID]/control	webrtc:///1/uuid/control
		Data	webrtc:///[VERSION]/[EndPointID]/data/[NODE_ID]/[PORT_ID]	webrtc:///1/uuid/data/0001/0000
WebAPI	URL	Management	https://[Authority]/[VERSION]/[EndPointID]/management/[API_NAME]	https://la.nicealliance.com/1/uuid/management/GetDeviceStatus
		Control	https://[Authority]/[VERSION]/[EndPointID]/control/[NODE_ID]/[API_NAME]	https://as.nicealliance.com/1/uuid/control/0001/GetStatus
		Data	https://[Authority]/[VERSION]/[EndPointID]/data/[NODE_ID]/[PORT_ID]	https://ds.nicealliance.com/1/uuid/data/0002/0001

4. Common Message Format

Data transferred over Management, Control and Data Interface shall be encapsulated into common message format in JSON. Common Message format consists of Header and Payload as follows.

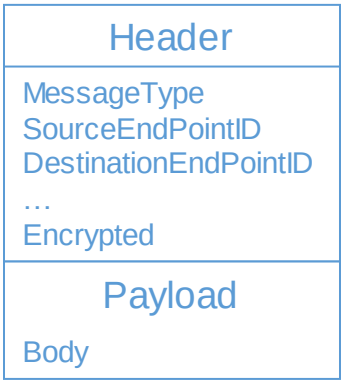

Figure 1. Common Message Format

4.1. Header structure

All JSON member are mandatory required.

Message	Key	Type	Size	Value	
Request	MessageType	String	8 (max)	request	Request message. Value is case-sensitive.
	SourceEndPointID	String	36	xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx	EndPointID of the caller. Format is defined in NICE

					Identifier Structure specification.
	DestinationEndPointID	String	36	xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxx	EndPointID of the callee. Format is defined in NICE Identifier Structure specification.
	DateTimeStamp	String	24	YYYY-MM-DDThh:mm:ss.sssZ	UTF date time sent the Request. Format is defined in NICE Date Time Format specification.
	CommandID	Integer			Number from 0. Caller must increment the number per sending the request message.
	CommandType	String	128 (max)	Example) 1/xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxx/control//0/StartStreaming	Refer to Destination ID.
	AccessToken	String			AccessToken compliant with JWT. The format is defined in NICE Authentication specification
	EncryptionOn	Boolean	1	1	Indicate that Body in Payload is encrypted. 1 must be set in the normal operation.
Response	MessageType	String	8 (max)	response	Response message. Value is case-sensitive.
	SourceEndPointID	String	36	xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxx	Same value as in the Request.
	DestinationEndPointID	String	36	xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxx	Same value as in the Request.
	DateTimeStamp	String	24	YYYY-MM-DDThh:mm:ss.sssZ	UTF date time sent the Response. Format is defined in NICE Date Time Format specification.

EncryptionOn	Boolean	1	1	Same value as in the Request.
ReplyID	Integer			Correspond to Command ID of Request.
ReplyStatusCode	Integer			API execution status which equals HTTP status code.
ReplyErrorCode	Integer			0 if ReplyStatus is "ok". The others are API specific status code which defined in each APIs.
ReplyStatusMessage	String	256 (max)		Detail information if Reply Status is NOT "ok". The value must be UTF-8 encoded string.

4.2. Payload Structure

Data encryption is strongly recommended in terms of end to end security.

Message	Key	Type	Size	Value	
Request / Response	Body	String		Base64 encoded string	Encrypted JSON Object which is defined in NICE Privacy and Security specification.

RequestMessage

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "title": "RequestMessage",
    "description": "Common Message Format",
    "additionalProperties": false,
    "properties": {
        "MessageType": {
            "type": "string",
            "default": "request"
        },
        "SourceEndPointID": {
            "type": "string"
        },
        "DestinationEndPointID": {
            "type": "string"
        },
        "DateTimeStamp": {
            "type": "string"
```

```
        },
        "CommandID": {
            "type": "integer"
        },
        "CommandType": {
            "type": "string"
        },
        "AccessToken": {
            "type": "string"
        },
        "EncryptionOn": {
            "type": "boolean",
            "default": "1"
        },
        "Payload": {
            "type": "object",
            "additionalProperties": true,
            "properties": {
                "Body": {
                    "type": "string"
                }
            }
        }
    },
    "required": [
        "MessageType",
        "SourceEndPointID",
        "DestinationEndPointID",
        "CommandID",
        "CommandType",
        "AccessToken",
        "DateTimeStamp",
        "EncryptionOn"
    ]
}
```

ResponseMessage

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "title": "ResponseMessage",
    "description": "Common Message Format",
    "additionalProperties": false,
    "properties": {
        "MessageType": {
            "type": "string",
            "default": "response"
        },
        "SourceEndPointID": {
            "type": "string"
        },
        "DestinationEndPointID": {
            "type": "string"
        },
        "DateTimeStamp": {
            "type": "string"
        },
        "ReplyID": {
            "type": "integer"
        },
        "ReplyStatusCode": {
```

```
            "type": "integer"
        },
        "ReplyErrorCode": {
            "type": "integer"
        },
        "ReplyStatusMessage": {
            "type": "string"
        },
        "EncryptionOn": {
            "type": "boolean",
            "default": "1"
        },
        "Payload": {
            "type": "object",
            "additionalProperties": true,
            "properties": {
                "Body": {
                    "type": "string"
                }
            }
        }
    },
    "required": [
        "MessageType",
        "SourceEndPointID",
        "DestinationEndPointID",
        "ReplyID",
        "ReplyStatusCode",
        "ReplyErrorCode",
        "ReplyStatusMessage",
        "DateTimeStamp",
        "EncryptionOn"
    ]
}
```

5. Transport Layer Security (TLS)

In order to achieve secure network transport the entities in NICE Eco System must support TLS as the underlying protocol of MQTT and HTTP.

5.1. Protocol Version

Protocol	Version	Support Level
SSL	1.0, 2.0, 3.0	Prohibited
TLS	1.0, 1.1	Prohibited
TLS	1.2	Mandatory
TLS	1.3 or later	Optional

5.2. Cipher Suites

TLS server and client is allowed to determine the priority of cipher suite according to their ability.

Cipher Suite Name	Key Exchange	Authentication	Encryption	MAC	PRF	Support Level
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDHE	ECDSA	AES-256-GCM	N/A	SHA384	Mandatory
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDHE	RSA	AES-256-GCM	N/A	SHA384	Mandatory
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDHE	ECDSA	AES-128-GCM	N/A	SHA256	Mandatory
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDHE	RSA	AES-128-GCM	N/A	SHA256	Mandatory
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	DHE	RSA	AES-256-GCM	N/A	SHA384	Mandatory

5.2.1. Key Exchange

PFS(Perfect Forward Security) is mandatory.

Less than 2048-bit of DH parameter must not use in case of DHE.

5.2.2. Authentication

Authentication is mandatory.

5.2.3. Encryption

128-bit or better security level and AEAD(Authenticated Encryption with Associated Data) must be supported. AES GCM is mandatory.

6. Datagram Transport Layer Security (DTLS)

DTLS is used by WebRTC protocol in NICE System.

6.1. Protocol Version

Protocol	Version	Support Level
DTLS	1.0	Prohibited
DTLS	1.2	Mandatory

6.2. Cipher Suites

DTLS cipher suites must be compliant with TLS's.

7. WebAPI

The NICE Web API is an interface for querying information from NICE Service or for sending information to NICE Device. Must utilize HTTPS and TLS when calling all methods.

There are two types of WebAPI in NICE.

- CMF(Common Message Format) API
- Management, Control and Data Interface

7.1. CMF (Common Media Format) API

This type of API is utilized in Management, Control and Data Interface. HTTP Request and Response contain the Common Message Format.

7.1.1. HTTP Method

When sending a HTTP POST or PUT, HTTP body must contain the Common Message. JSON object which is defined in NICE specifications may be set to the Body field of Common Message in order to send API parameter or receive API response.

Content-Type header must be set to "application/json". URL encoded body is not supported.

7.1.2. Request

All requests must contain Common Message (MessageType is "request") and the following HTTP header.

- Content-Type: application/json
- Authorization: Bearer ***AccessToken***
    - AccessToken must be set to the Authorization HTTP header. It is not allowed to send as part of the query string in URI.

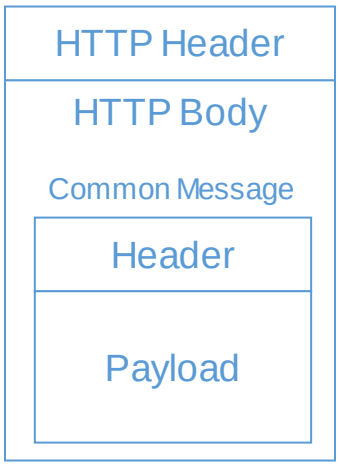

Figure 2. HTTP Message Structure

7.1.3. Responses

All responses must contain Common Message (MessageType is "response") and the following HTTP header.

- Content-Type: application/json
- X-Content-Type-Options: nosniff

7.2. Restful API

This type of API may used other than Management, Control and Data interface.

Regarding third party specific WebAPI for their own service, it does not need to be compliant with this spec.

7.2.1. Method

GET, POST, PUT, PATCH, DELETE are allowed.

7.2.2. Request

HTTP header must be compliant with the following rules.

- In case of sending HTTP request containing URL-encoded data, Content-type header must be set as follows and key/value pairs are set according to RFC3986.
    - Content-Type: application/x-www-form-urlencoded
- In case of sending HTTP request containing JSON in the request body, Content-Type must be set as follows.
    - Content-Type: application/json
- In case that AccessToken is required, it must set as follows.
    - Authorization: Bearer ***AccessToken***.

7.2.3. Response

HTTP header must be compliant with the following rules.

- In case of sending HTTP response containing JSON in the response body, Content-Type must be set as follows.
    - Content-Type: application/json
- X-Content-Type-Options: nosniff

7.3. HTTP status

The HTTP response status codes are issued by a server to reply a client's request. Each Status-Code is listed below.

Status Code	Message	Description
200	OK	Request successful
201	Created	The request has been fulfilled and resource created. A URI for the created resource is returned in the Location header.
202	Accepted	The request has been accepted for processing, but processing is not yet complete.
400	Bad Request	The request is malformed, such as HTTP body format error.
401	Unauthorized	Authentication failed.
403	Forbidden	Authentication succeeded but the client doesn't have permission to the request resource.
404	Not Found	When a non-existent resource is requested.
406	Unacceptable	The client presented a content type in the Accept header which is not supported.
405	Method Not Allowed	The error for an unexpected HTTP method.
413	Payload too large	The request size exceeded the given limit.
415	Unsupported Media Type	The requested content type is not supported.
429	Too Many Requests	The error is used when there may be DOS attack detected or the request is rejected due to rate limiting.
500	Internal Server Error	An unexpected condition prevented the server from fulfilling the request.
501	Not Implemented	The requested operation is not implemented.
503	Service Unavailable	Temporarily unable to process the request.

7.4. How to use the WebAPI

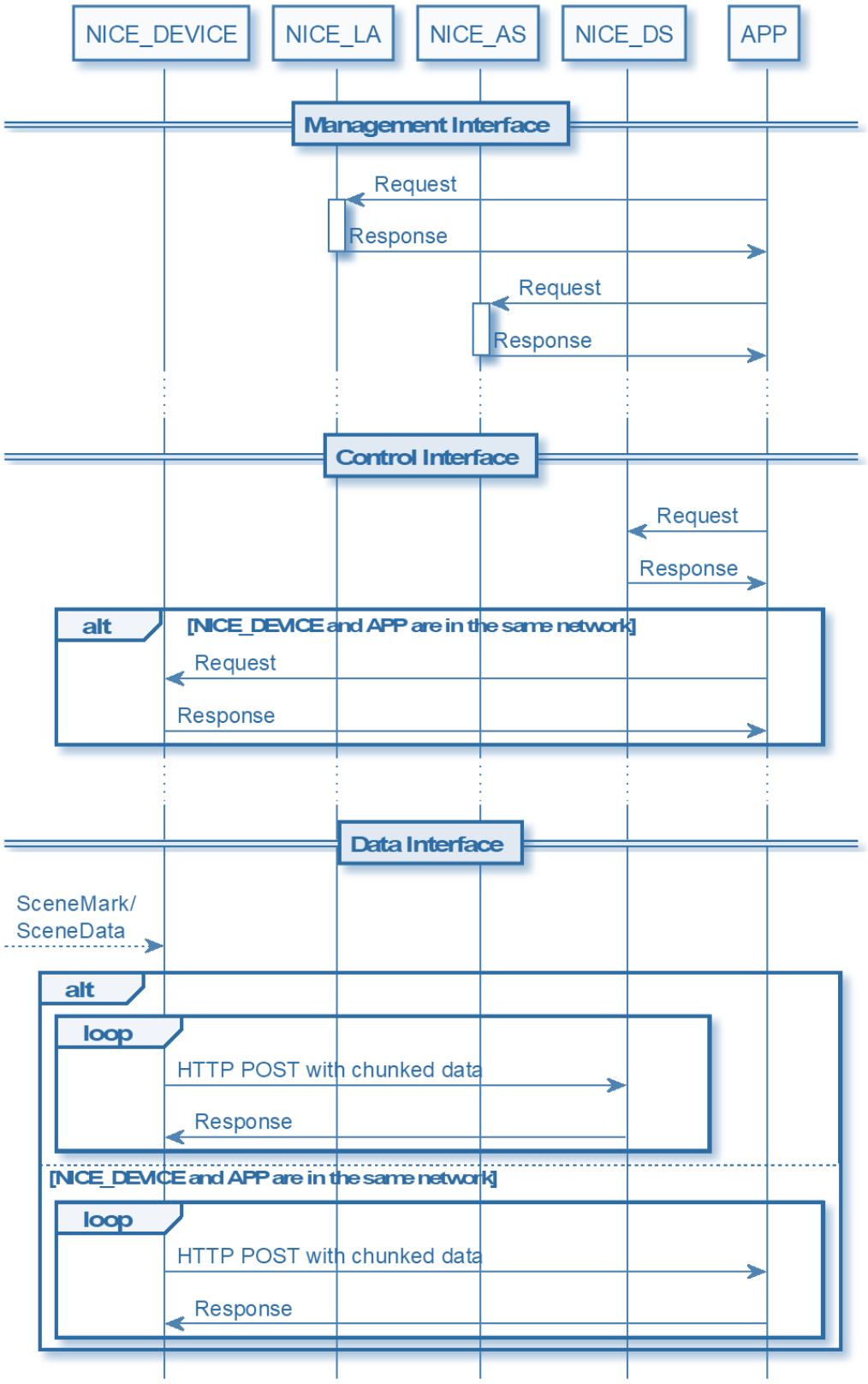

Figure 3. How to use the WebAPI

8. MQTT

8.1. Overview

MQTT is bi-directional messaging protocol. NICE System uses MQTT for Management, Control and Data Interface. In order to ensure the network connectivity, MQTT over WSS (WebSocket over TLS) must be supported.

Management and Control over MQTT is Request and Response style. Therefore receiver must send back the Response message in reaction to the Request message from sender. The message is defined in Common Message Format.

In case that sender has not received the Response, the message might have been lost by some reasons. Sender may need the resend in such a case.

8.2. QoS

The Quality of Service (QoS) level is an agreement between the sender and the receiver of a message that defines the guarantee of delivery, and MQTT defines 3 levels of QoS.

- 0 : At most once
- 1 : At least once
- 2 : Exactly once

NICE Publisher and Subscriber need to set QoS level according to the Interface as the following.

Interface	Publisher	Subscriber
Management	0	0
Control	0	0
Data	Set by SceneMode Object	Set by SceneMode Object

For MQTT on Data Connections QoS levels 0, 1 and 2 shall be supported.

8.3. Set Up of MQTT Connection

NICE Device acquires the broker for Management Interface from NICE LA by WebAPI request on the first power up. Device establishes MQTT connection and subscribe the topic of the Device to receive Management command.
Once the Device is assigned to the End User account and configured by SetManagementObject, the Device is bound to the NICE Account Service. The Device must establish another MQTT connection with the NICE Account Service Broker and subscribe to the topic of the Device to receive Management command.
If the Device is released from the NICE Account Service, it returns to the NICE License Authority as a default, and the Device must not connect to NICE Account Service Broker any more.

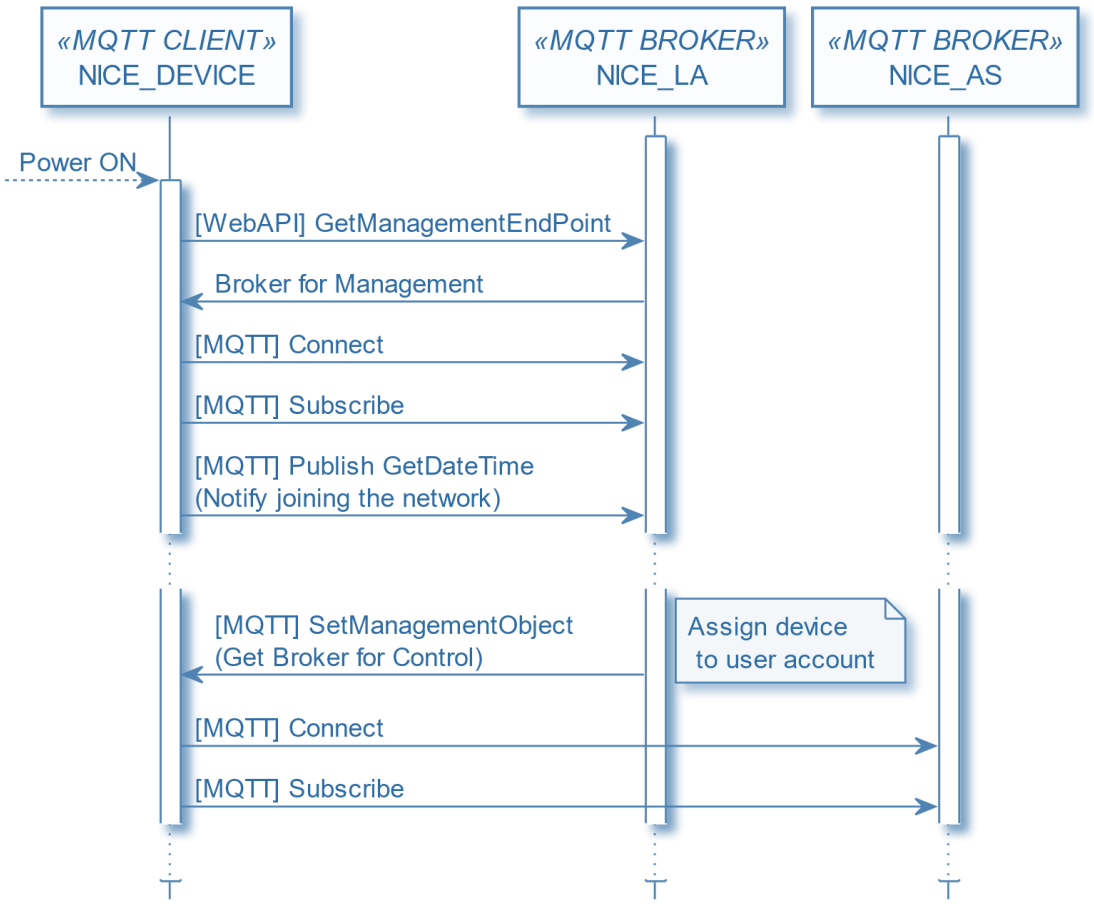

Figure 4. Setup MQTT Connection

NICE Device needs to know two topics - the topic to subscribe for receiving the Request, and the destination topic for sending Response.
The topic name for Management and Control Interface are provided by GetManagementEndPoint and SetManagementObject respectively.
The destination topic name of Data Interface is provided by SetSceneMode.
Refer to Destination ID regarding the topic name definition.

8.4. How to use MQTT

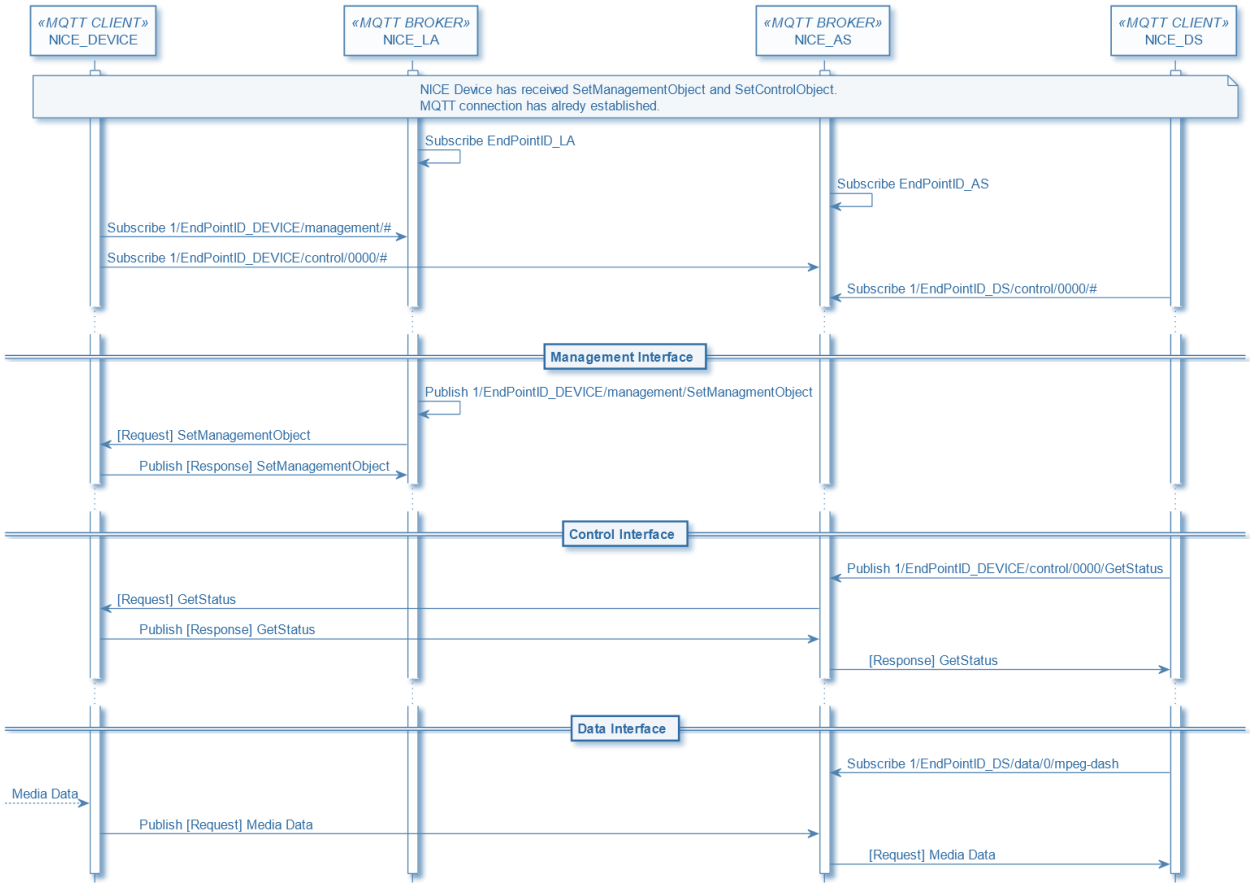

Figure 5. How to use MQTT

9. WebRTC

9.1. Set Up of WebRTC connection

The following diagram shows the life cycle of WebRTC in NICE System. Typically PEER1 is NICE Device, and PEER2 is NICE App or Service.

For more information about EstablishControlSession, CloseControlSession and WebRTCSignaling, refer to NICE App/Service Specification.

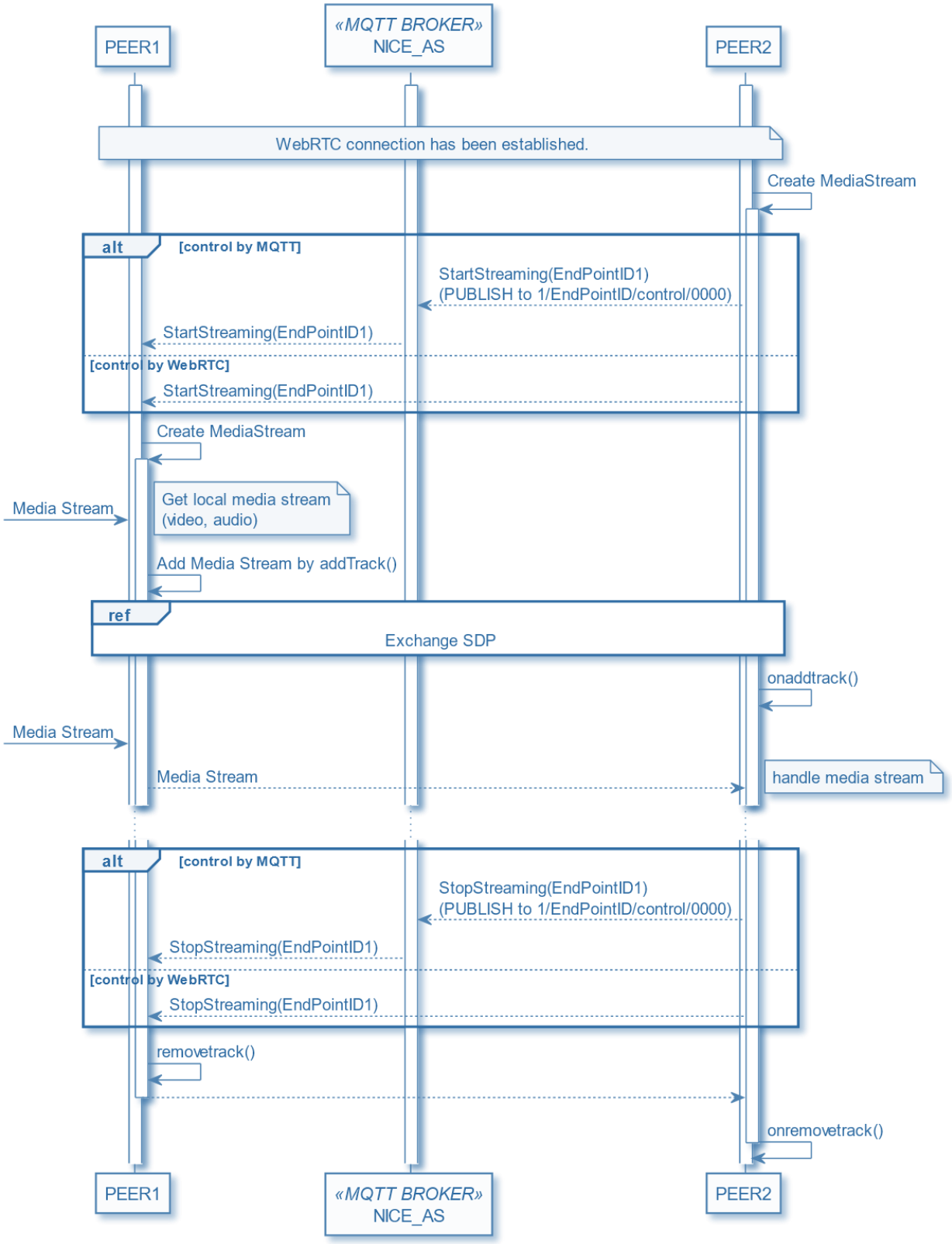

Figure 6. The life cycle of WebRTC in NICE System

9.2. Signaling Protocol

Signaling Protocol is not defined in WebRTC specification. NICE utilize MQTT for the signaling protocol to establish WebRTC connection. Therefore MQTT Management must be working prior to initiate WebRTC.

The source peer attempting to connect with the destination sends EstablishControlSession to NICE AS. Destination peer receives SetControlObject and initiates WebRTC protocol. Peer must subscribe the following MQTT topics in order to exchange SDP. MQTT topic which the destination peer should subscribe, is specified in the command payload.

Peer	MQTT Topic to subscribe	
PEER1	[VERSION]/EndPoint1/management/WebRTCSignaling	To receive Answer SDP. EndPointID1 is EndPointID of PEER1. Topic name is specified in the message of SetControlObject.
PEER2	[VERSION]/EndPoint2/management/WebRTCSignaling	To receive Offer SDP. EndPointID2 is EndPointID of PEER2.

9.3. Peer Connection

The following is general steps to establish WebRTC Peer Connection in NICE. Typically PEER1 is NICE Device and PEER2 is App/Service.

1. PEER2 send EstablishControlSession to NICE AS to initiate WebRTC connection with PEER1.
2. PEER1 receive SetControlObject command published by NICE AS. The message of the command contains the topic name of PEER2.
3. PEER1 initiate WebRTC connection protocol.
    1. PEER1 create WebRTC object by RTCPeerConnection().
    2. PEER1 create DataChannel object for control by createDataChannel(). The label of the DataChannel is "[VERSION]/EndPointID1/control", which is the topic name of PEER1's control. (See Destination ID)
    3. PEER1 create SDP1(offer) and set it as Local Description.
    4. PEER1 initiate ICE gathering.
    5. Signaling protocol (offer SDP) by MQTT
        1. PEER1 set the SDP1 to the MQTT message and publish it to the topic of PEER2.
        2. PEER2 receive SDP1 by MQTT message.
    6. PEER2 create WebRTC object by RTCPeerConnection().
    7. PEER2 set SDP1 and set it as Remote Description.
    8. PEER2 create SDP2(answer) and set it as Local Description.
    9. PEER2 receive the DataChannel handle (which label is "[VERSION]/EndPointID1/control").
    10. PEER2 initiate ICE gathering.
    11. Signaling protocol (answer SDP) by MQTT
        1. PEER2 set the SDP2 to the MQTT message and publish it to the topic of PEER1.
        2. PEER1 receive SDP2 by MQTT message.
    12. PEER1 set SDP2 as Remote Description.
4. Peer Connection is established.
    1. ICE Agent keeps alive the established connection.

2. DataChannel for control is available.
3. SceneMark/SceneData Streaming is ready.
4. Media Streaming is ready.

5. Close Peer Connection.
    1. PEER1 or PEER2 publish CloseControlSession command to NICE AS.
    2. PEER1 receive SetControlObject command published by NICE AS.
    3. PEER1 close() the WebRTC object.
    4. PEER2 close() the WebRTC object if receive the event by onicecandidate().

Note that Trickle ICE method is preferable. It would reduce the time of ICE gathering.

9.4. Data Channel

The peers which attempt to connect each other by WebRTC must support Data Channel.

Data Channel for Control must be created after Peer Connection is established if the Device is configured that WebRTC is utilize for Control. NICE Device must support the Control command according to the Device capability.

Data(e.g. SceneMark, SceneData,...) in Data Interface must be sent by Data Channel if the Device is configured for Data over WebRTC. NICE Device must support multi-channel to send different types of data simultaneously. NICE Device shall create Data Channel for each type of data. Refer to Destination ID regarding the label name of DataChannel.

Data delivery option of Data Channel must be configured as "ordered" and "reliable".

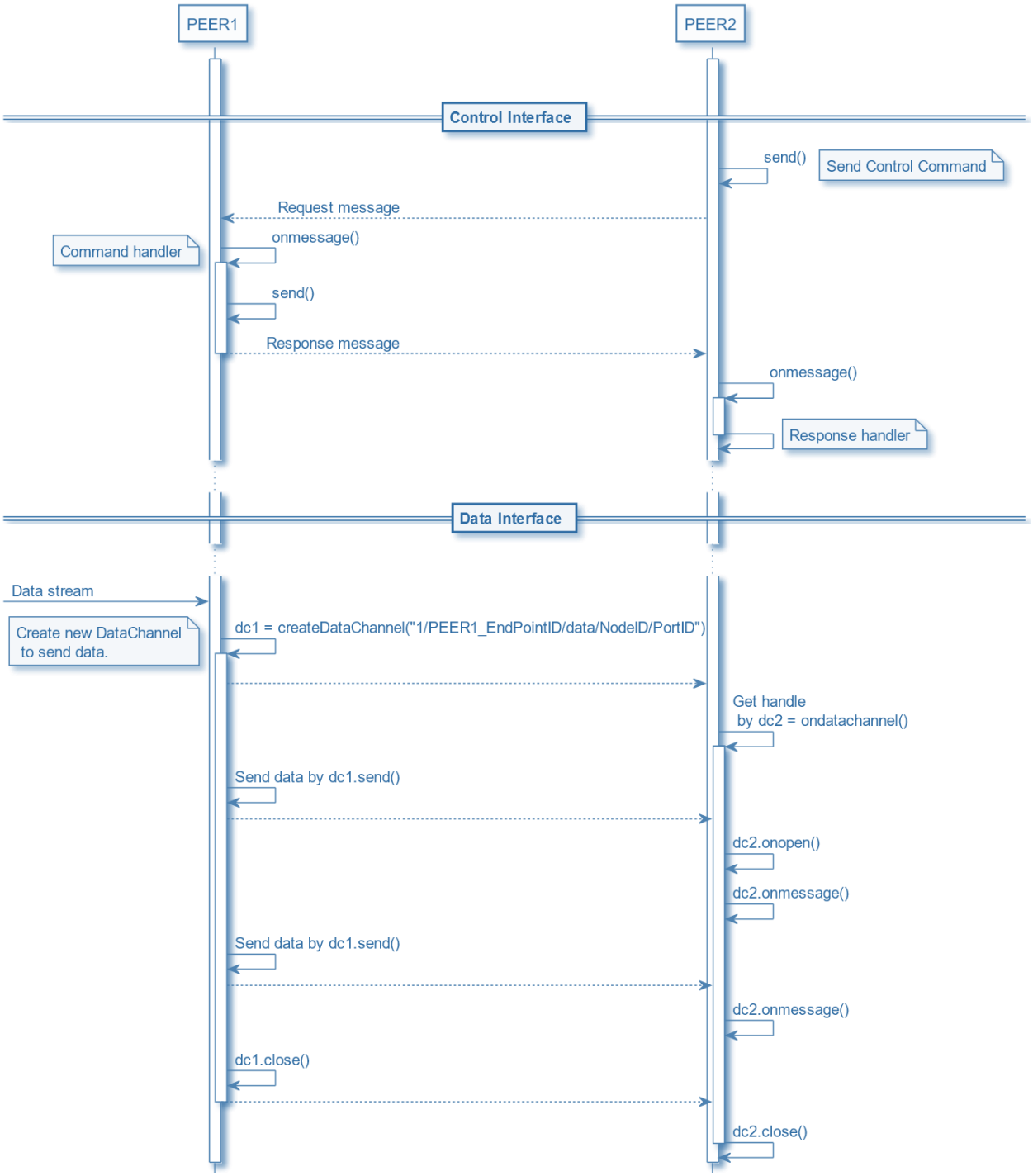

Figure 7. Data Channel

9.5. Live Streaming

Live Streaming is used to send media(audio or/and video) with low latency from NICE Device to App/Service by using WebRTC. The following diagram shows overview of Live Streaming, where PEER1 is NICE Device and PEER2 is App or Service.

Live Streaming is triggered by PEER2 by StartStreaming command of control.

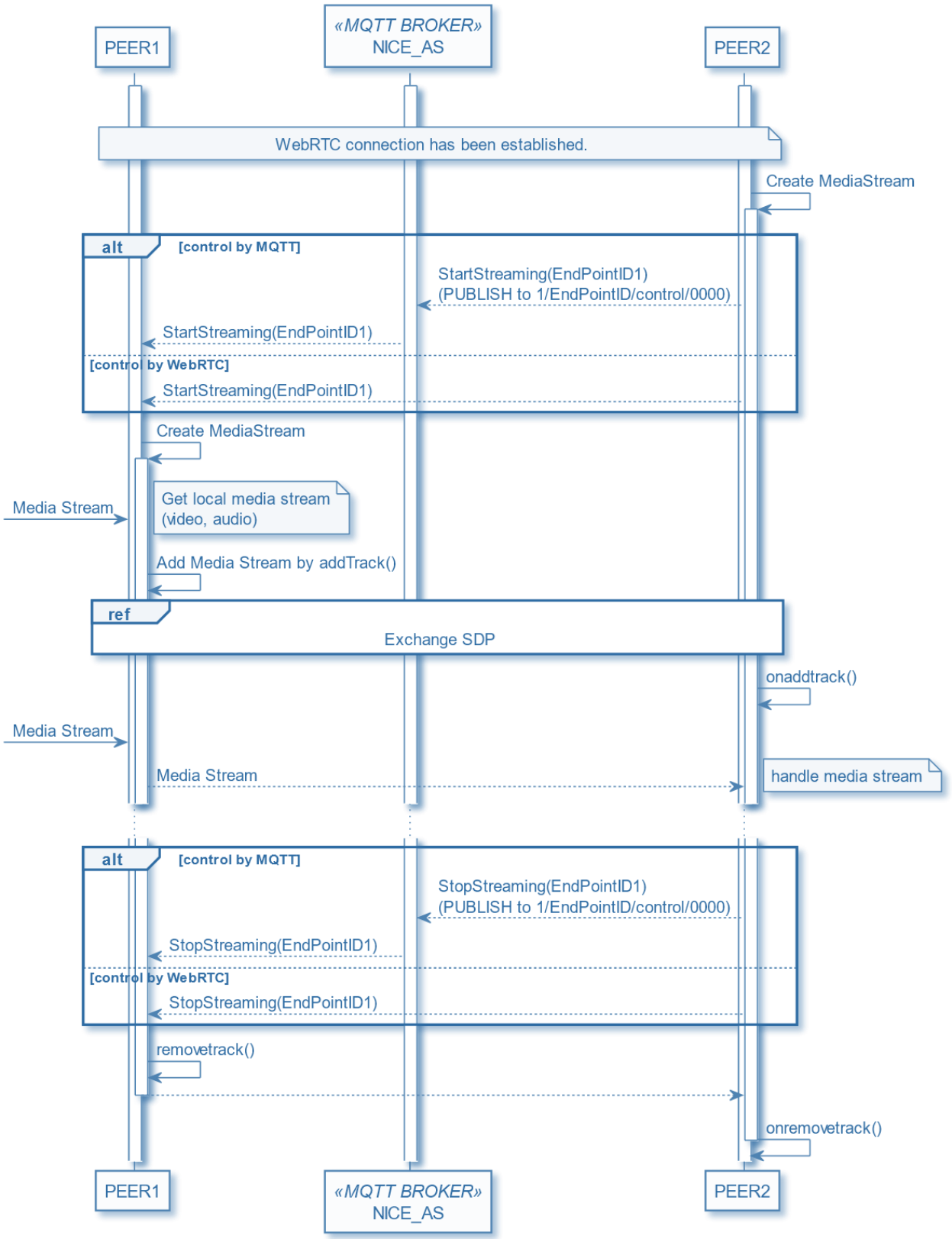

Figure 8. Live Streaming