**Agenda Item: 10.5**

**Source: Samsung Electronics Co., Ltd. (Rapporteur)**

**Title: iRTCW Permanent Document**

**Version: 0.60**

**Document for: Discussion & Agreement**

This document includes texts, figures, or other information that may complement TS 26.113 or be included later. Sections 4-8 of this document are related to clauses 4-8 of TS 26.113 respectively.

**Introduction**

**4. System description**

**4.1 High-level architecture**

**4.2 iRTC client in terminal**

**4.3 Web real-time communication**

**5. Functional components**

**5.1 Audio**

**5.1.1 Microphone description**

A microphone array description is provided in [10] and further extended below as an example. The sending or receiving iRTC client, or audio infra may identify the direction of each microphone from a set of angles and a type (e.g., pick up pattern). Alternatively, they may identify the position of each microphone from a set of cartesian coordinates.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In many spatial audio formats, a microphone processing block converts raw signals from microphones into a device-independent representation.

Diagram

Description automatically generated

The angles are expressed in units of 1/10000 radians. For example, 3.1416 (π) radians is expressed as 31416.

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Unit** | **Definition** | **Note** |
| Yaw (𝛹, psi) | int | Direction angle | -31416 < 𝛹 ≦ 31416 |
| Pitch (𝛳, theta) | int | Elevation angle | -31416 < 𝛳 ≦ 31416 |
| Roll (𝛷, phi) | int | Rotation angle | -31416 < 𝛷 ≦ 31416 |
| MicrophoneType | int | A number that uniquely identifies microphone type | May be used for indicating vendor-defined types |

These parameters can describe the orientation of each microphone relative to a camera, and its type. The rotation angle, not defined in [10], can be used to indicate the rotation of a directional microphone whose pick up pattern is not axisymmetric. It is assumed that the RGB or D cameras for capturing 2/3D video are located on a plane parallel to the ZY plane, pointing objects in the direction of X axis. This information on the direction of microphones (with respect to cameras) can be used for improved spatial capture of the sound field and subsequent post processing in terms of noise reduction or sound source decomposition.

Although typical UEs (at the time of writing this contribution) have 2-3 microphones, they are rarely located on the same plane. Usually, a microphone is located on top of a display while another is located at the bottom of UE. Sometimes a microphone is located on the other side of display (with main cameras). Therefore, the Linear or Planar mode [10], designated with the wMicArrayType parameter, would not be applicable.

In some contemporary UEs, e.g., foldable devices, the direction or elevation angle may be different in each session or even vary during a session, and the sending or receiving iRTC client, or audio infra may need to be informed of such changes for proper rendering. In addition to the geometric description, if applicable, other or additional information may be beneficial to the microphone processing, e.g.:

* acoustic overload point (AOP) for estimating the loudness (pressure level) of sounds in recorded audio signals
* total harmonic distortion (THD) for understanding nonlinearity in captured audio signals
* microphone sensitivity response for potential post equalisation.

Editor’s Note: Need for transmitting this information will be discussed and studied further.

**5.2 Video**

**5.2.1 3D video capture**

**5.2.1.1 Depth formats & point cloud generation**

Resolutions and rates of RGBD frames would influence the overall quality and their minimum required (or recommended) values may be specified for iRTC & other RTC applications. FoVs (horizontal and vertical) of RGBD frames, typically depending on the number and location (distance) of sensors, may also be specified.

**5.2.1.2 Depth formats & point cloud generation**

Depth information is typically stored in 16-bit unsigned integer format, which indicates the distance between each point and a vertical plane including the depth sensors, which typically consist of an (IR) transmitter and one or more receivers, in millimeter [1], [2]. Each point in 3D space is mapped onto a 2D image plane via a series of transforms illustrated below.

Diagram

Description automatically generated

Mapping of 3D points to 2D image plane [3]

In the figure, [**R** **t**]T represents the rotation and translation from a 3D world coordinate system to a 3D camera’s coordinate system, whose parameters can be supplied by UE’s motion sensors. K is the camera intrinsic matrix defined as

Shape

Description automatically generated with medium confidence

where

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Unit** | **Definition** | **Note** |
| *fx* | float | X-axis focal length (in pixel) |  |
| *fy* | float | Y-axis focal length (in pixel) |  |
| *cx* | float | X-axis principle point (in pixel) |  |
| *cy* | float | Y-axis principle point (in pixel) |  |
| *s* | float | Skew coefficient | Zero if image axes are perpendicular |

Depending on notation, transpose of **K** is also used as a camera intrinsic matrix. These device-specific information is supplied by typical OSs used in UEs [4], [5], which assume s=0. Therefore, a point cloud in e.g., PLY format can be generated from a pair of RGB and D frames by reversing the transforms, with parameters available to applications in UEs. The FoVs of RGB and D cameras will be in general different and how to align them is left to the discretion of the implementation.

**5.2.2 Size measurement of 3D Objects**

In [6], the needs for scaling 3D objects were illustrated for several scenarios. The conference managing server or receiving iRTC client needs the size information of 3D objects for a proper scaling. The size of a 3D object captured with a visual sensor (i.e., video camera), can be achieved with the help of the capture device information (focal length and sensor size) and the (estimated) distance to the subject.

This means to estimating a physical size of an object (or user), first the image size of the object is determined in the captured image data, and secondly the relation between the image size and the physical size is determined with the help of the camera metadata (i.e., focal length) and the objects distance to the capture device (e.g., based on a depth sensor or machine learning estimate). The resulting object size metadata comprises the size of the object to enable a rendering device or server to establish the “actual” size of the virtual object in the virtual environment in accordance with its physical size of the object in physical space.

Additionally, there are smartphone applications that estimate the length, width and height, or volume of objects seen through the viewfinder. The measurement in general depends on the types of cameras used. These applications typically place a box or cube around an object and show the measured values. If the cube around a 3D object is set to a minimum, then the size of the cube may be used as an estimate of the object’s size, as illustrated below:

Graphical user interface

Description automatically generated with medium confidence

Camera-based measurement applications [7], [8]

Considering that typical depth cameras represent the depth information in millimeter [9], the size information may be similarly represented as

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Unit** | **Definition** | **Note** |
| *w* | Int | Width of cube (in mm) |  |
| *h* | Int | Height of cube (in mm) |  |
| *d* | int | Depth of cube (in mm) |  |

The size information can be signalled to a far-end iRTC client or conference managing server for scaling the 3D object to other objects or backgrounds. The size information may be transmitted periodically or in an on-demand fashion, depending on applications, and may also be used locally.

Graphical user interface, diagram, text, application

Description automatically generated with medium confidence

The error in the size estimation may be reduced by following common rules for placing boxes or cubes during initial measurements. A cube may not always be an optimal form to contain a point cloud captured with a limited number of RGB or D sensors of UEs, which may also depend on the FoVs of the cameras.

Editor’s Note: This will be discussed also with Video SWG.

**5.2.3 Use cases and requirements for volumetric video dynamic 3D representation**

Use cases 19 and 22 in TR 26.998 provide an AR conferencing experience. In both cases, participants can capture their 3D dynamic representation in real-time and share it with others in a shared AR experience. In this clause we state the requirements related to capturing and transporting the dynamic 3D representation of an object under conversational constraints. Dynamic 3D representation in this context is a format for defining the 3D model of an object (e.g., a caller) that is captured in real-time using one or more cameras. A dynamic 3D representation can be a point cloud, 3D mesh, or similar representation (e.g., RGB and depth).

TR 26.998 section 4.4.5 defines some existing formats used for dynamic 3D representations. The Virtual Reality Industry Forum (VRIF) has recently issued their first Volumetric Video Guidelines, addressing volumetric video production workflows and media profile standards for volumetric media distribution [11]. Based on the requirements for encoding, transporting, and decoding volumetric media, the following sub-categories are defined for the different use cases and their associated requirements.

**5.2.3.1 AR two-party calls over IMS**

Graphical user interface, diagram

Description automatically generated

The proposed solution uses RTP for low-latency delivery of dynamic 3D representations.

The use case in the above figure [xx] establishes a bidirectional AR two-party call, which may or may not use the MCU (MRF for IMS). It should be possible to combine other functions, e.g., 2D video, 360-degree video, images, etc., which are not shown in the figure for simplicity.

The dynamic 3D representations delivery in this case is over RTP. It is bidirectional with conversational latency requirements. The dynamic 3D representation can be delivered over one of the two paths shown in the figure. Path A goes through the MCU (MRF for IMS) and Path B is point-to-point.

Non-real-time 3D representations can be delivered via the data channel as shown in the figure.; the term 3D representation here includes both dynamic and static 3D representations, which are not captured and delivered under conversational latency requirements. The data channel is a WebRTC data channel or an IMS data channel. The IMS data channel used is as defined by TS 26.114. Further requirements, if needed, can be defined for using the data channel to transport 3D representations.

The following set of requirements relate to the bidirectional conversational dynamic 3D representation:

1. Call setup and control: this building block covers
   1. signaling to setup a call or a conference – basic functions already provided by MTSI and will be covered also in IRTCW.
   2. fetching of the entry point for the AR experience. The protocol needs to support upgrading and downgrading to/from an AR experience. Dependency on MeCAR to define device types.
2. Formats: The media and metadata types and formats include in addition to the ones already covered by MTSI, volumetric media. Format properties and codecs need to be defined for dynamic 3D representations along with appropriate RTP payload formats and functions. Appropriate codecs need to be defined by MeCAR for encoding, decoding and rendering dynamic 3D representations.
3. Real-time encoding and decoding with latency requirements for conversational media.
4. Enhancements to SDP, scene description to support AR telephony.
5. For AR telephony media types (e.g., dynamic 3D representation), the necessary QoS characteristics need to be defined.
6. Support for AR media processing in the MCU (MRF for IMS).
7. 5G system integration: offering the appropriate support by the 5G system to AR telephony includes:
   1. discovery and setup of MCU (MRF) resources to process AR telephony media types.
   2. defining the necessary QoS characteristics for AR telephony media types.
   3. data collection and reporting.

**5.2.3.2 AR multi-party calls**

A screenshot of a computer

Description automatically generated with medium confidence

The use case in the above figure establishes a multiparty call. The call may be unidirectional (one dynamic 3D representation sender and multiple receivers, 1:N) or bidirectional (multiple dynamic 3D representation senders and multiple dynamic 3D representation receivers, N:N). The 1:N case can be addressed first as part of this work as it is simpler. The N:N case can be addressed later.

In addition to the requirements of use case 2.1, the following requirements need to be considered for AR multi-party calls:

* Signalling for establishing a multiparty call. This may be done in a similar way as for traditional MTSI/WebRTC calls.
* Expanding scene description to address the case of multiple senders and multiple receivers; defining appropriate procedures to maintain position of all participants in the rendered space for each participant.
* Mixing/transcoding in the MCU (MRF for IMS) to combine content from multiple participants. This may include, e.g., scaling and placement of 3D representations in a virtual room. Other requirements can also be studied.
* More advanced requirements may also be considered based on the existing use cases in FS\_5GXR and FS\_5GSTAR
  + Integration with other 5G services such as 5GMS for DASH delivery of AR media (that is not used for delivering conversational AR media but possibly video streams for a shared experience) along with conversational media.
  + Maintaining consistent head motion and eye-contact in a multiparty call with 3D avatars.

Editor’s Note: The discussion here uses entities, such as MRF, from the IMS architecture. The figures can be modified once the architecture for WebRTC has been defined.

**5.2.3.3 AR conferencing**

The proposed solution uses cloud/SFU architecture for AR calling (e.g., AR conferencing use-case 19 in TR 26.998). The solution should use the media capabilities / APIs defined in AR device architecture in TR 26.998 and TS 26.119.

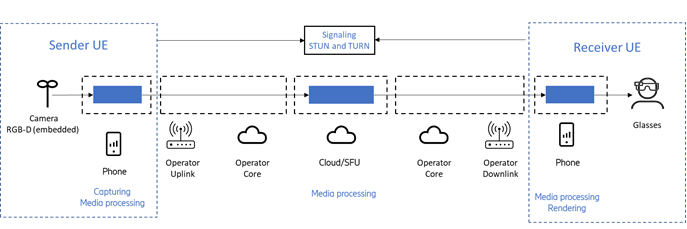
The workflow is described in the following steps:

* Capturing device is equipped with 3D video capture capabilities e.g., RGB and depth sensors.
* Captured media streams, e.g., compressed RGB and depth, are transmitted via secure WebRTC channel to the cloud.
* The cloud media processing unit processes the received streams to generate a mesh or point cloud representation.
* The cloud SFU (selective forwarding unit) handles communication between multiple device instances.
* The receiving device decodes and renders the received media stream.

The architecture supports the transmission of multiple media streams such as audio, RGB and depth streams (human representation), and non-dynamic 3D object representation.

The media transport between sender UE and cloud, and the cloud and receiver UE can use WebRTC data and/or media channel.

The solution uses WebRTC signalling and discovery mechanisms for establishing communication between different peers.



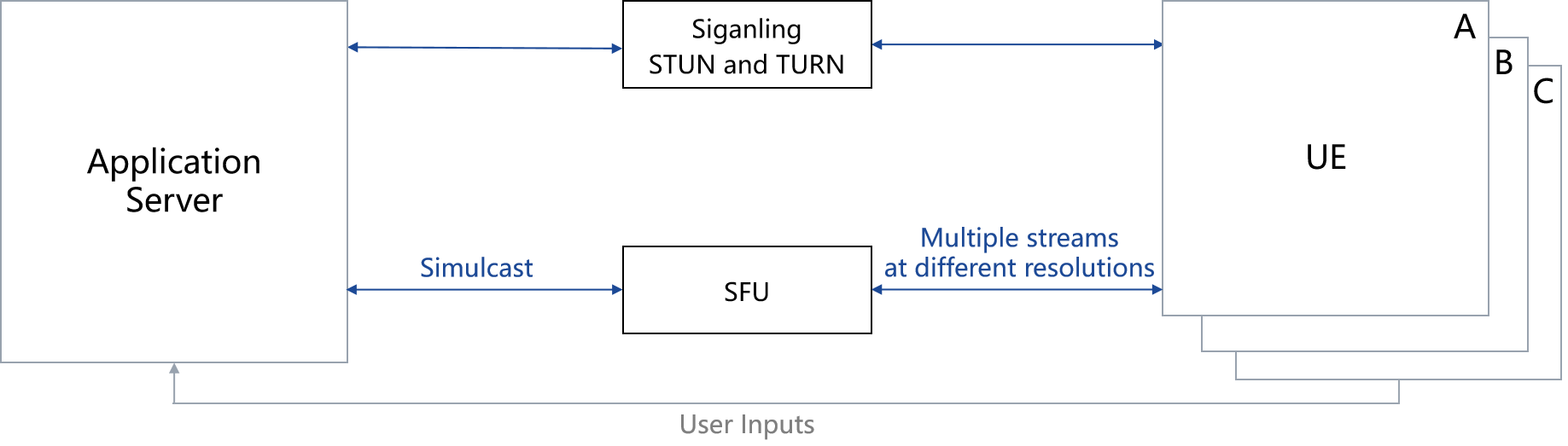
**5.2.3.4 APIs for AR conferencing**

TR 26.998 defined several real-time communication use-cases. Use cases 19 and 22 in TR 26.998 refer to AR conferencing scenarios. Section 5.2.3.3 (AR conferencing) describes an AR Conferencing use-case. A list of APIs is described based on the device architecture specified in MeCAR (TS 26.119).

|  |  |  |  |
| --- | --- | --- | --- |
| **MeCAR Function** | **Mapping to use-case (AR conferencing)** | **List of APIs** | **Consumer of the API** |
| Media Access Functions (MAF) | APIs for uplink media transmission such as compressed RGB and depth, 3D mesh or point cloud.  APIs for downlink media transmission such as point cloud, 3D mesh, RGB-D, or 2D video (split rendering). | * Authentication between devices (e.g., using tokens) * Create a session between device and server * Transmit uplink streams using publish or producer patterns * Connect to the session – both peers (sender and receiver) * Receive downlink streams using subscribe or consumer patterns * Download 3D objects, scene description, or other data | Environment can be browser or game engine or stand-alone app (Windows, iOS, Linux) for sender (camera client). Environment is game engine for receiving data, because at the moment this is no browser support for glasses yet.  MAF is tightly coupled with WebRTC libraries/SDKs. |

Editor’s Note: API parameters will be discussed and studied further.

**5.2.3.5 XR streaming over WebRTC**



The workflow is described in the following steps:

* The Application Server shall be GPU-capable, such as a virtual machine provided by a custom cloud-hosted platform. It runs the game logic, renders every frame in real-time, and continuously encodes (e.g., H.264, VP 8, VP 9 video compression) the rendered video frames along with the audio into a media stream.
* The SFU (selective forwarding unit) receives the stream from the Application Server and deliver it to the recipients peers (typically connected web browsers), optionally subsetting the data (e.g. lower bitrates, resolutions, or framerates) to adapt to the prevailing network conditions of each recipient peer.
* The receiving devices display the received media stream, and send the user’s inputs back to the Application Server using WebRTC data channel. The user’s inputs can be generic (e.g., keyboard, mouse, touch events) or App specific.

**5.2.3.6 AR call solution for smartphones or tablets**

The use case in the above figure establishes a bidirectional AR call. In this case, the 5G modem and AR media processing are integrated in the device. The following requirements need to be considered:

* Signalling for establishing a bidirectional call. It can be done in a similar way as for traditional MTSI/WebRTC calls.
* 2D video frames acquisition. The local UE can acquire the rendered video frames by using commercially available AR frameworks.
* Real-time video format conversion. The rendered AR video frames are converted into a standard YUV video format in real time before sending out to the remote UE.
* The converted video frames can be sent to the remote UE via WebRTC media channel. WebRTC/IMS data channel can be used for exchanging user’s input such as keyboard, mouse, touch events, and other custom events (Optional).

Graphical user interface

Description automatically generated

**5.2.3.6.1 AR video frames acquisition**

The rendered AR video frames can be acquired from AR Runtime by the following steps:

1. The AR Application is launched.
2. The AR application creates AR Session in the AR Runtime.
3. The AR application communicates with the scene manager to provide a scene description either through an application interface or through a well-defined scene description document, possibly retrieved over the network.
4. Based on the scene requirements, the capabilities of the AR Runtime, as well as the capabilities of the media access function, a set of media pipelines in the uplink and downlink are established.
5. The media access function accesses the network resources or sends data to the network using the established media pipelines.
6. The AR Runtime captures the relevant AR data to match the device capabilities, and uses Graphic Engine (OpenGL ES, Vulkan, Metal, DirectX, etc) to render the AR objects on the camera feed.
7. The AR Runtime presents the rendered AR video frame locally, and simultaneously sends the rendered AR Video Frames to the Video Format Conversion module.

**5.2.3.6.2 Video format conversion**

**5.2.4 3D video compression**

The 3D video data can be directly compressed, or analysed and converted to codes for animating an avatar [6]. As an avatar trained for a person can represent only the very person, volumetric-types can be used to represent generic 3D objects or arbitrary people.

**5.2.4.1 Volumetric-type**

There are standards such as V3C that can be used for compressing point cloud [21]. V3C can also be used for compressing input from one or more RGB-D cameras. The extent of support for V3C levels and profiles during Release-18 will be discussed and defined as part of MeCAR. A generic V3C pipeline to be considered for the RTC use cases follows.

The following figure shows the sender pipeline for delivery of V3C encoded video. The input from RGB-D cameras is first pre-processed. The pre-processing may include processing the depth map, conversion to point clouds (if V-PCC is used) and creation of raw video and associated metadata, i.e., atlas data, etc. The pre-processed data is then passed to the encoders:

* The video encoder for encoding the video components, such as, geometry, occupancy attribute, and,
* The atlas encoder for the atlas data (metadata).

The video components can be frame packed to create a single video Elementary Stream. The figure shows the case where the video components are kept separate; hence, multiple signals flow from the video encoder to the RTP encapsulation. The encapsulated bitstreams are then sent to the receiver.

Diagram

Description automatically generated

Sender pipeline for V3C content delivery

The following figure shows the receiver pipeline for V3C content. The one or more video bitstreams for the video components are fed to a video decoder. The atlas component is decoded using an atlas decoder that parses the atlas data to retrieve the required information for reconstructing the volumetric video from the 2D video components. The post processing may include conversion of the decoded video component from one representation to the other, e.g., from YUV 4:2:0 to RGB format. The content is then rendered based on the pose of the receiver device.

Diagram, schematic

Description automatically generated

Receiver pipeline for V3C content delivery

The grey boxes in the figures show aspects that are relevant for the work in 3GPP. Note that the pre-processing aspects will be for information, if included. Whether any aspects of post-processing and rendering are to be included as normative will be evaluated and defined as part of MeCAR.

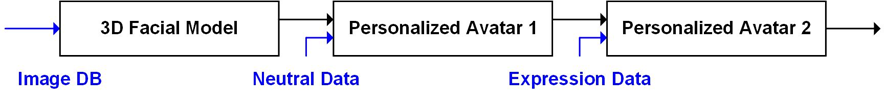
An internet draft for the payload format of V3C is under development in IETF. The draft defines the payload of V3C atlas sub-bitstreams and provides information on how to deliver it with other V3C atlas video-bitstreams as separate or bundled RTP streams. The RTP payload format for V3C video sub-bitstreams is based on the video codec used to encode the V3C video component. The viability of V3C for real-time delivery of volumetric video in AR conversational use cases is discussed in [26].

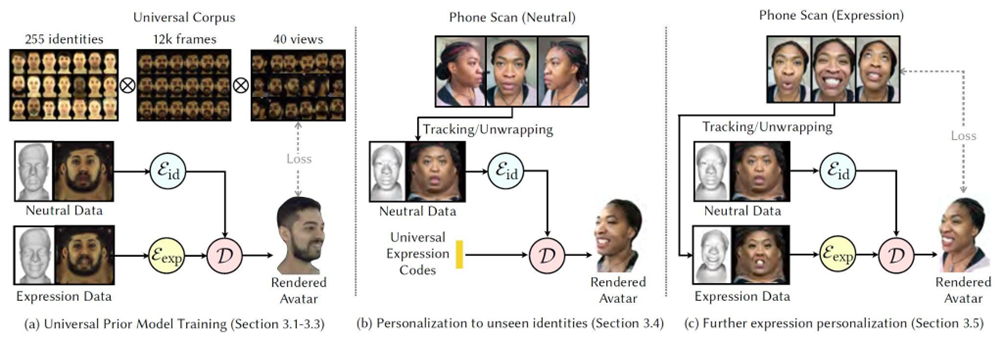
NOTE: The figures provide a simple pipeline based on MTSI pipelines. If needed they can be matched with the XR client architecture based on the agreed way forward in MeCAR, IBACS and iRTCW.

**5.2.4.2 Avatar-type**

Use of an avatar needs prior work for generating a 3D facial model of a person to represent.

There are many approaches to generate and train 3D avatars. From a large amount of image or video data, a 3D model for facial areas or heads can be defined, which is trained with neutral (no expression) and a diversity of expression data captured from a person.





3D avatar generation process [22]

A set of data professionally-captured for building 3D facial models can be downloaded from [23]. In [22], the neutral and expression data were captured with a smartphone having an RGBD camera. Note that the display is used to guide the series of expressions to make.

Once an avatar is trained, it is shared by the sender and receiver devices. During conversations or conferences, movement of facial areas are captured and converted to codes that are to animate a 3D avatar, and transmitted with compressed audio. It was shown that a VR headset could drive multiple avatars simultaneously [24].

A movie explaining the generation and operation of avatars based on [22] can be watched in [25].

**5.3 Sensor**

**5.4 Transport protocols**

**5.4.1 Real-time interaction metadata transport over data channel**

The interaction metadata may be carried in WebRTC data channel [15] given the data size and low-latency requirements.

**5.4.1.1 Real-time interaction metadata relevant use cases**

TS 26.928 [20] defines core use cases and scenarios for XR, and the interaction metadata relevant use cases are as followings:

1. Real-time XR sharing such as real-time 3D communication, AR animated avatar call, and 5G shared spatial data. The relevant metadata such as gesture, position, FOV, viewport, facial expression may be used to render the view, control the animated avatar visual and auditory appearance, The XR clients may also continuously send sensing data to a cloud service which provides the map back to client.
2. XR multimedia Streaming such as immersive 6DoF streaming and emotional streaming. The relevant metadata such as user pose, FOV, viewport and head or body motion may be used to control immersive 3DoF+ or 6DoF views from different position and angles and help the users to navigate through the scene.
3. Online XR gaming, the relevant metadata such as body movement, viewport and gaming control metadata are essential for the rendering and reaction to enable presence.
4. XR conference such as 360-degree conference meeting, 3D shared experience, 6DoF VR conferencing and XR meeting. The relevant metadata such as pose, FOV, viewport are used to control the view or poster rendering in 360-degree or 6DoF conferencing.

**5.4.1.2 WebRTC data channel**

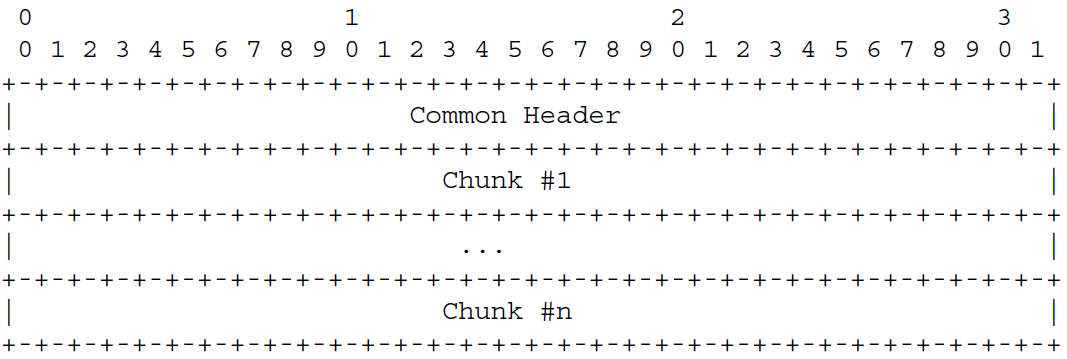
In the WebRTC framework, communication between the parties consists of media (for example, audio and video) and non-media data. Media is sent using the SRTP, and non-media data is handled by using the Stream Control Transmission Protocol (SCTP) [16].

The priority associated with a media flow or data flow is classified as “very-low”, “low”, “medium”, or “high” in the API. WebRTC implementations may attempt to set QoS on the packet sent according to the guidelines in [17].

The SCTP provides the following features for transporting non-media data between browsers:

* Support of multiple unidirectional streams
* Ordered and unordered delivery of user messages
* Reliable and partial reliable transport of user message

An SCTP packet is composed of a common header and chunks. A chunk contains either control information or user data. The SCTP packet format is shown below:



SCTP packet format

The common header field includes a source port number, a destination port number, a verification tag and checksum, as shown in the following figure.

Table

Description automatically generated with medium confidence

SCTP common header format

Each chunk is formatted with a chunk type field, a chunk-specific Flag field, a Chunk Length field, and a value field as shown below.

Table

Description automatically generated

SCTP chunk field format

SCTP defines a number of chunk types, and one of the types is Payload Data (DATA) when chunk type is 0. The payload data format is shown in the following figure. The U bit indicates unordered data chunk; B and E bit indicate the fragmented user message; the payload protocol identifier represents an application specific protocol identifier and is passed to SCTP by its upper layer and sent to its peer.

Calendar

Description automatically generated

SCTP chunk payload data type format

**5.4.1.3 Potential solution for using data channel for interaction metadata**

The WebRTC data channel may be used to convey real-time interactive metadata.

[Editor’s Note: carrying interaction metadata over the Data Channel using this approach may be useful but further study of use cases and other types of interaction metadata will be investigated to identify other potential solutions.]

It is proposed to use registered SCTP PPID 53, “WebRTC Binary” [15], to carry the interaction metadata. A generic payload format is shown in the following figure.



Generic data channel payload format for the timed metadata

The 32-bits Subprotocol payload ID indicates the subprotocol or specifications used for the metadata payload data format, such as OpenXR.

The 16-bits Metadata type field may indicate the metadata type specified in the payload subprotocol.

[Editor’s note: the data length may be extended if 16-bits is not sufficient to accommodate the potential metadata type indication such as URN]

The 16-bit Metadata attributes field may indicate the metadata attributes such as time synchronization and reliability.

A WebRTC data channel may be optionally mapped by means of an SDP offer/answer [18] or DCEP [19] procedure to a “subprotocol” parameter indicating which protocol the client expects to exchange data via the channel. This “subprotocol” parameter value is registered for WebRTC data channels as per [19] Section 9.1, or equivalently [19], Section 5.1. If the “subprotocol” parameter is not present, then its value defaults to an empty string.

For the interaction metadata specified in MeCAR, the subprotocol could be defined as “3gpp-timedmetadata”, and the message format would be binary. The specific syntax and semantics of the interaction metadata are indicated by the 32-bits subprotocol payload ID field in the User Data of the SCTP packet.

The following figure is an example of an SDP offer that adds 1 timed metadata data channel stream (stream id=110).



Example SDP offer with timed metadata signalling

The following figure is an example considering further down the syntax and semantics of the generic candidate fields introduced in the figure for the case of an OpenXR XR\_EXT\_hand\_tracking metadata:

* 32-bits subprotocol payload ID is set to an OpenXR ID value.
* 16-bits metadata type field is set to 52 which is the registered extension number for XR\_EXT\_hand\_tracking in OpenXR.
* 16-bits metadata attributes field may indicate the metadata attributes such as time synchronization and reliability.
* OpenXR defines a 64-bit signed integer representing nanoseconds (XrTime) and associates it to the APIs. Yet, SCTP does not carry timestamp information. For the payload protocol that does not define timestamp, a timestamp field may be added to the chunk user data section when a metadata timestamp attribute bit, T, is set to 1 as below.
* OpenXR (XrTime) 64-bit signed timestamp in nanoseconds.



Example of OpenXR XR\_EXT\_hand\_tracking metadata payload format

**5.4.1.4 Security considerations**

The interaction class real-time metadata can contain sensitive information tracking the interactions of an end user, e.g., elements of pose, tracking information of palm, hand, or face, as well as controller inputs. Therefore, the integrity and confidentiality of metadata in transit is in some scenarios, depending on the application requirements, necessary.

The transport of the real-time interaction class metadata over the WebRTC data channel and in particular over SCTP as of clause 2.3 ensures by default, [15], integrity and confidentiality given the underlying DTLS over UDP secure transport.

**6. Session management**

**6.1 WebRTC functions in 5GS**

**7. Inter-working**

**8. Packet-loss handling**

**9. Architecture and function**

**9.1 WebRTC QoS architecture**

Collaboration scenarios.

**9.1.1 5G support for OTT WebRTC**

In this scenario, the application provider offers a WebRTC service to their customers and is responsible for the security and privacy of the data exchanged over their service. The application provider desires to improve the quality of the service for mobile users.

The collaboration scenario is similar to the 5GMS streaming model, where all involved Application Servers (AS) are in non-trusted domain.

The following figure depicts this collaboration scenario:

A screenshot of a computer

Description automatically generated with medium confidence

* + 1. **MNO-provided trusted WebRTC functions**

In this collaboration scenario, the user utilizes a variety of WebRTC-based conferencing services for personal and work purposes. The user is concerned about their privacy, e.g., through threats from man-in-the-middle attacks that result from usage of untrustworthy ICE functions. the user opts for using a trusted WebRTC configuration, that is provided by their MNO, which offers ICE functions such as STUN, TURN, MCU, etc.

In addition, the MNO will offer the required traffic handling to the WebRTC sessions, from some or all application providers, based on existing SLAs or based on user contractual agreements.

The following figure depicts this collaboration scenario:

A screenshot of a computer

Description automatically generated with medium confidence

* + 1. **MNO-facilitated WebRTC services**

The MNO may offer several WebRTC-based services such as remote gaming, AR telephony, etc. These services are facilitated and managed by the MNO and are offered to the MNO’s customers exclusively.

All WebRTC functionality is hosted by the MNO, which ensures the quality of the services through appropriate network assistance, such as QoS allocation.

The following figure depicts this collaboration scenario:

A picture containing text, electronics

Description automatically generated

* + 1. **Inter-operable WebRTC services**

As an extension of previous collaboration scenario, a globally inter-operable WebRTC service is provided, where mobile users from different MNOs are able to join the same service and benefit from the 5G system support for better end-to-end quality of service. In this collaboration scenario, the WebRTC functions are hosted by one or more MNOs.

The following figure depicts this collaboration scenario:

Shape, arrow

Description automatically generated

**9.2 WebRTC functions in the context of iRTCW**

**9.2.1 General**

We have agreed on 4 different collaboration scenarios for iRTCW at the SA4-118e meeting. The collaboration scenarios are listed here for convenience:

1. 5G support for OTT WebRTC: in this scenario the WebRTC session runs completely over the top. However, the MNO may offer support in form of QoS allocation, bitrate recommendations, and QoE report collection based on request by the UE.
2. MNO-provided trusted WebRTC functions: in this scenario the MNO offers trusted support functions such as ICE servers to the WebRTC application on the UE.
3. MNO-facilitated WebRTC services: the MNO may host and facilitate WebRTC sessions by providing a trusted WebRTC signaling server, which may also offer 5G network assistance.
4. Inter-operable WebRTC services: collaboration scenario 3 is extended with functions to support MNO to MNO inter-operability.

Based on the documented collaboration scenarios, we identify the following functions and describe their roles.

**9.2.2 Potential 3GPP-defined functions**

**9.2.2.1 General**

These functions will be discussed and possibly confirmed in the context of the architecture that will be defined by the 5G\_AREA WI.

**9.2.2.2 Provisioning server**

The provisioning server may enable an application provider to perform provisioning of the following functionalities:

* QoS support provisioning for WebRTC sessions
* Charging provisioning for WebRTC sessions
* Collection of consumption and QoE metrics data provisioning related to WebRTC sessions
* Offering ICE functionality provisioning such as STUN and TURN servers
* Offering WebRTC signaling servers provisioning, potentially with interoperability to other signaling servers.

The provisioning server may not be relevant to all collaboration scenarios and some of the 5G support functionality may be offered without application provider provisioning.

**9.2.2.3 Configuration server**

The configuration server stores WebRTC-related configuration information and makes them accessible to the UE. It stores information and recommendations to operate network-assisted WebRTC sessions over 5G.

The configuration information may consist of static information such as the following:

* Recommendations for media configurations
* Configurations of STUN and TURN server locations
* Configuration about consumption and QoE reporting
* Discovery information for WebRTC signaling and data channel servers and their capabilities.

**9.2.2.4 Media Session handler (MSH)**

The MSH is an entity running on the UE, which assists with the 5G integration of the WebRTC application. It exchanges, on behalf of the application, information about the WebRTC sessions with the network.

The MSH receives information about a new WebRTC session from the application. It relays the information to the Support Function. It also receives events and other network information about the WebRTC session from the Support Function, which it may relay to the application.

**9.2.2.5 Network support function**

The support functionality includes the following:

* Network Support Function receives information about a WebRTC session and its state
* Network Support Function requests QoS allocation for a starting or modified session
* Network Support Function receives notification about changes to the QoS allocation for the ongoing WebRTC session
* Network Support Function exchanges information about the WebRTC session with the trusted STUN/TURN/Signaling Server, e.g. to identify a WebRTC session and associate it with a QoS template.

**9.2.3 WebRTC functions**

**9.2.3.1 Trusted ICE functions**

The MNO may offer trusted ICE functions to the WebRTC application to be used during the WebRTC ICE gathering phase. These functions may be STUN and TURN servers that facilitate NAT and Firewall traversal.

The MNO-operated trusted ICE functions may assist with the 5G integration of the WebRTC application. This could be done by triggering network assistance to starting or ongoing WebRTC sessions.

**9.2.4 iRTCW-defined functions**

**9.2.4.1 Trusted WebRTC signaling server**

The trusted WebRTC signaling server is used to setup and manage MNO-operated WebRTC applications. They offer a standardized signaling protocol for the session setup to both parties of the WebRTC session. The WebRTC signaling server will handle the offer/answer exchange and will have access to the SDP in both directions.

The WebRTC signaling server may use that knowledge to offer network assistance and other 5G features to the endpoints of the WebRTC session.

**9.2.4.2 Inter-working function**

This function provides inter-working functionality to enable MNO-facilitated WebRTC sessions that involve end-points across different MNOs. They may for example provide cross-network signaling functionality to allow WebRTC signaling server that are hosted in different networks to communicate, in order to establish and manage the WebRTC sessions.

**9.2.4.3 Trusted media server**

A media server may be offered by the MNO to support WebRTC sessions. It may offer a wide range of functionality such as:

* a content server that serves content to the WebRTC application, e.g. through a data channel
* media processing functionality: may be used by the WebRTC application as a relay that performs some media processing function such as transcoding, recording, 3D reconstruction, etc.
* scene composition functionality: the server may compose a 3D scene and distribute it to several point-to-point WebRTC sessions
* MCU functionality: the server may offer multi-party conferencing functionality to merge a number of point-to-point WebRTC sessions
* SFU (Selective Forwarding Unit) functionality: the server may offer the selection, copy, and forwarding functionality of IP steams produced by multiple WebRTC endpoints (i.e., participants).

**9.2.5 Mapping to collaboration scenarios**

The following table provides an initial mapping of the identified functions to the collaboration scenarios:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Functions/CS | **Collaboration scenario 1** | **Collaboration scenario 2** | **Collaboration scenario 3** | **Collaboration scenario 4** |
| **Provisioning server** | Optional | Optional | Optional | Optional |
| **Configuration server** | Optional | Required | Required | Required |
| **MSH** | Required | Optional | Optional | Optional |
| **Network support function** | Required | Required | Optional (maybe fulfilled by WebRTC signaling server) | Optional |
| **Trusted ICE function** | N/A | Required | Optional | Optional |
| **Trusted WebRTC signaling server** | N/A | N/A | Required | Required |
| **Trusted media server** | N/A | Optional | Optional | Optional |

**10. Considerations on RTC APIs**

**10.1 ProvisionedConfiguration resource**

The data model for the ProvisionedConfiguration resource is specified in Table 10.1-1 below:

Table 10.1-1: Definition of ProvisionedConfiguration resource

| Property name | Data Type | Cardinality | Description |
| --- | --- | --- | --- |
| offerTrustedStunServers | boolean | 0..1 | Indicates if the AF should provide a list of trusted STUN servers to the UE for usage with RTC sessions of this application provider. |
| stunServers | array(URL) | 0..1 | An array of trusted STUN servers that the application can use as ICE candidates. |
| offerTrustedTurnServers | boolean | 0..1 | Indicates if the RTC AF should provide a list of trusted TURN servers to the UE for usage with RTC sessions of this application provider. |
| turnServers | array(URL) | 0..1 | An array of trusted TURN servers that the application can use as ICE candidates. |
| offerTrustedSwapServers | boolean | 0..1 | Indicates if the AF should provide a list of trusted SWAP servers to the UE for usage with RTC sessions of this application provider. |
| swapServers | array(URL) | 0..1 | An array of trusted WebRTC signaling servers that support the SWAP protocol. If provided, the application shall use one of the listed servers for RTC sessions of this application provider. |

**10.2 PolicyTemplate resource**

The data model for the PolicyTemplate resource is specified in table 10.2‑1 below:

Table 10.2-1: Definition of PolicyTemplate resource

| Property | Type | Cardinality | Usage | Description |
| --- | --- | --- | --- | --- |
| policyTemplateId | ResourceId | 1..1 | C: RO R: RO U: RO | Identifier of this Policy Template assigned by the 5GMS AF that is unique within the scope of the Provisioning Session. |
| state | string enum | 1..1 | C: RO R: RO U: RO | A Policy Template may be in the PENDING, INVALID, READY, or SUSPENDED state.  Only a Policy Template in the READY state may be instantiated as a Dynamic Policy Instance and applied to media streaming sessions. |
| stateReason | Problem‌Details | 1..1 | C: RO R: RO U: – | Additional details about the current state of this Policy Template exposed to the 5GMS Application Provider by the 5GMS AF.  The instance sub-property shall be present and shall indicate the URL of this Policy Template resource.  The title sub-property shall be present and shall indicate a human-readable representation of the state property specified above, e.g. "Policy Template ready for use" or "Policy Template invalid".  The detail sub-property shall be present and shall indicate a human-readable status/error message.  All other properties shall be omitted. |
| externalReference | string | 1..1 | C: RW R: RO U: RW | Additional identifier for this Policy Template, unique within the scope of its Provisioning Session, that can be cross-referenced with external metadata about the media streaming session. |
| qoSSpecification | M1‌QoS‌Specification | 0..1 | C: RW R: RO U: RW | Specifies the network quality of service to be applied to media streaming sessions at this Policy Template. |
| rtcQosSpecification | array(RTCQoSSpecification) | 0..1 | C: RW R: RO U: RW | Specifies the network quality of service to be applied to the different media streams of the RTC session. |
| application‌Session‌Context | Object | 1..1 |  | Specifies information about the application session context to which this Policy Template can be applied. |
| sliceInfo | Snssai | 0..1 | C: RW R: RW  U: RW | As defined in clause 5.4.4.2 of TS 29.571 [12]. |
| dnn | Dnn | 0..1 | C: RW R: RW  U: RW | As defined in clause 5.3.2 of TS 29.571 [12]. |
| charging‌Specification | Charging‌Specification | 0..1 | C: RW R: RW  U: RW | Provides information about the charging policy to be used for this Policy Template. |

The *RTCQoSSpecification* object is defined in table 10.2-2 below:

Table 10.2-2: Definition of RTCQoSSpecification object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Property name | Data type | Cardinality | Usage | Description |
| serviceDataFlowDescription | ServiceDataFlowDescription | 1..1 |  | The 5-Tuple that identifies the service data flow for which the QoS dynamic policy is requested. |
| mediaIdentifier | String | 1..1 |  | Provides an identifier for the media stream to associate with the corresponding service component in the QoS Policy. |
| marBwDlBitRate | BitRate | 1..1 |  | Maximum requested bit rate for the Downlink. |
| marBwUlBitRate | BitRate | 1..1 |  | Maximum requested bit rate for the Uplink. |
| minDesBwDlBitRate | BitRate | 0..1 |  | Minimum desired bit rate for the Downlink. |
| minDesBwUlBitRate | BitRate | 0..1 |  | Minimum desired bit rate for the Uplink. |
| mirBwDlBitRate | BitRate | 1..1 |  | Minimum requested bit rate for the Downlink. |
| mirBwUlBitRate | BitRate | 1..1 |  | Minimum requested bandwidth for the Uplink. |
| desLatency | Integer | 0..1 |  | Desire Latency. |
| desLoss | Integer | 0..1 |  | Desired Loss Rate. |
| pduSetMarking | PDUSetMarking | 0..1 |  | A description of the PDU Set and End of Burst marking configuration for the session.  In release-18, this shall only be present in the case of an RTP stream. |

The *PDUSetMarking* object is defined in table 10.2-3 below:

Table 10.2-3: Definition of PDUSetMarking object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Property name | Data type | Cardinality | Usage | Description |
| serviceDataFlowDescription | ServiceDataFlowDescription | 1..1 |  | The 5-Tuple that identifies the service data flow for which the QoS dynamic policy is requested. |
| headerExtensionVersion | Integer | 1..1 |  | The RTP header extension version. |
| localIdentifier | Integer | 1..1 |  | A unique identifier of the RTP header extension in the scope of the media session. |
| format | Boolean | 0..1 |  | Indicates if a short or a long header extension format is used. When set to false, a short 1-byte header extension format is being used. |
| pduSetSizeActive | Boolean | 0..1 |  | A flag to indicate if the PDU Set size in bytes is present in the RTP header extension. |

**10.3 MetricsReportingConfiguration resource**

The Metrics Reporting Provisioning API allows a 5G-RTC System operator or a 5G-RTC Application Provider to configure the Metrics Collection and Reporting procedure for real-time media streaming.

The metrics reporting provisioning procedure is as defined in clause 7.8 of TS 26.512.

The data model for metrics reporting provisioning API defined in clause 7.8.3 of TS 26.512 can be extended for 5G-RTC media services. The extended MetricsReportingConfiguration resource is specified in Table 10.3-1 below with the changes highlighted:

Table 10.3-1: Definition of MetricsReportingConfiguration resource

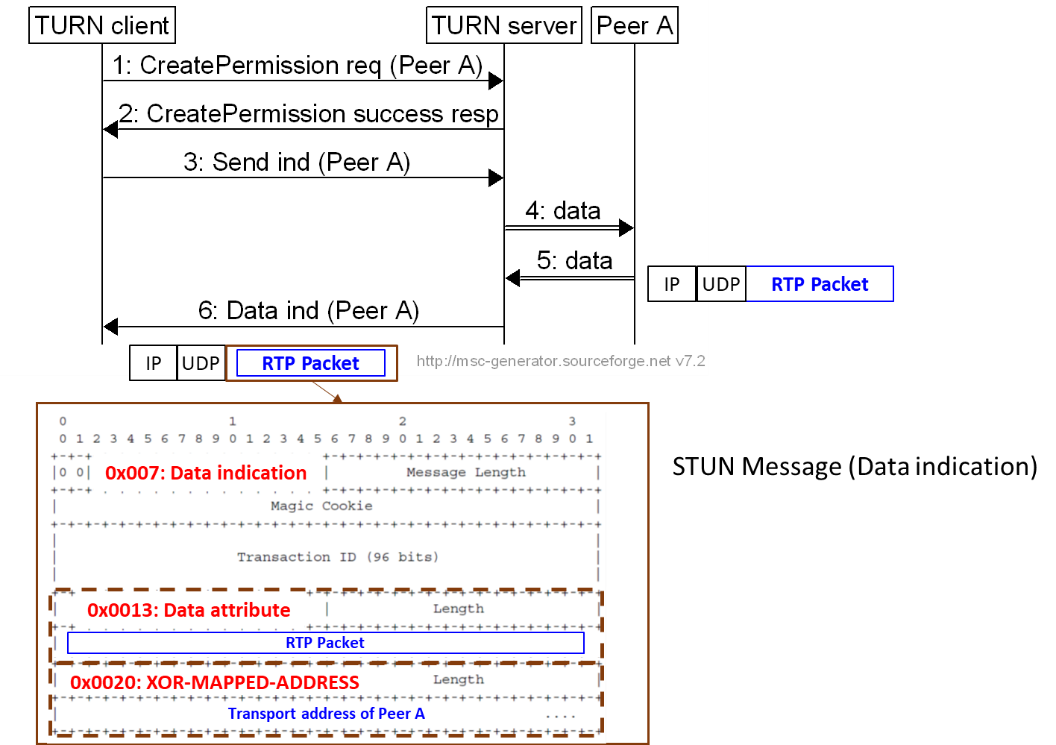
| Property name | Type | Cardinality | Description |
| --- | --- | --- | --- |
| *metricsReportingConfigurationId* | ResourceId | 1..1 | An identifier for this Metrics Reporting Configuration assigned by the 5GMS AF that is unique within the scope of the enclosing Provisioning Session. |
| *scheme* | Uri | 0..1 | The scheme associated with this Metrics Reporting Configuration. A scheme may be associated with 3GPP or with a non-3GPP entity.  For RTC media streaming, if not specified, the 3GPP metrics scheme urn:‌3GPP:‌ns:‌PSS:‌RTC:‌QM1 defined in clause 5.1.3 shall apply. |
| *dataNetworkName* | Dnn | 0..1 | The Data Network Name (DNN) which shall be used when sending metrics reports.  If not specified, the default DNN shall be used. |
| *reportingInterval* | DurationSec | 0..1 | The time interval between successive metrics reports. The value shall be greater than zero.  If not specified, a single final report shall be sent after the media streaming session has ended. |
| *samplePercentage* | Percentage | 0..1 | The proportion of media streaming sessions for which metrics shall be reported, expressed as a floating-point value between 0.0 and 100.0.  If not specified, reports shall be sent for all sessions. |
| *urlFilters* | array(String) | 0..1 | A non-empty list of Media Entry Point URL patterns for which metrics shall be reported.  If not specified, reporting shall be done for all media streaming sessions initiated within the scope of the parent Provisioning Session. |
| *samplingPeriod* | DurationSec | 1..1 | The time interval the RTC Client should wait between sampling the QoE metrics specified by this metrics reporting configuration. *SamplingPeriod* value shall be equal to or less than the *samplingDuration* value. |
| *samplingDuration* | DurationSec | 0..1 | The time duration specified in the media stream for which the QoE metrics will be reported by the RTC Client. There shall be only one range per measurement specification. If the " *samplingDuration*" field is not present, the metrics range shall be for the whole session duration. |
| *metrics* | array(String) | 0..1 | If present, a non-empty list of metrics which shall be collected and reported.  In the case of RTC media streaming and for the 3GPP scheme urn:‌3GPP:‌ns:‌PSS:‌RTC:‌QM1 the listed metrics shall correspond to one or more of the metrics as specified in clauses 9.1.2, and the quality reporting scheme and quality metric reporting protocol as defined in clauses 5.1.3 and 9.1.3, respectively, shall be used to produce and send metrics reports.  Metrics related to virtual reality media, as specified in TS 26.118 clause 9.3, may also be listed in the metrics configuration, and shall be reported according to the quality reporting scheme defined in clause 9.4 of TS 26.118.  If omitted, the complete (or default, as applicable) set of metrics associated with the specified scheme shall be collected and reported. |

**11. PDU Set Marking with TURN**

**11.1 TURN data delivery**

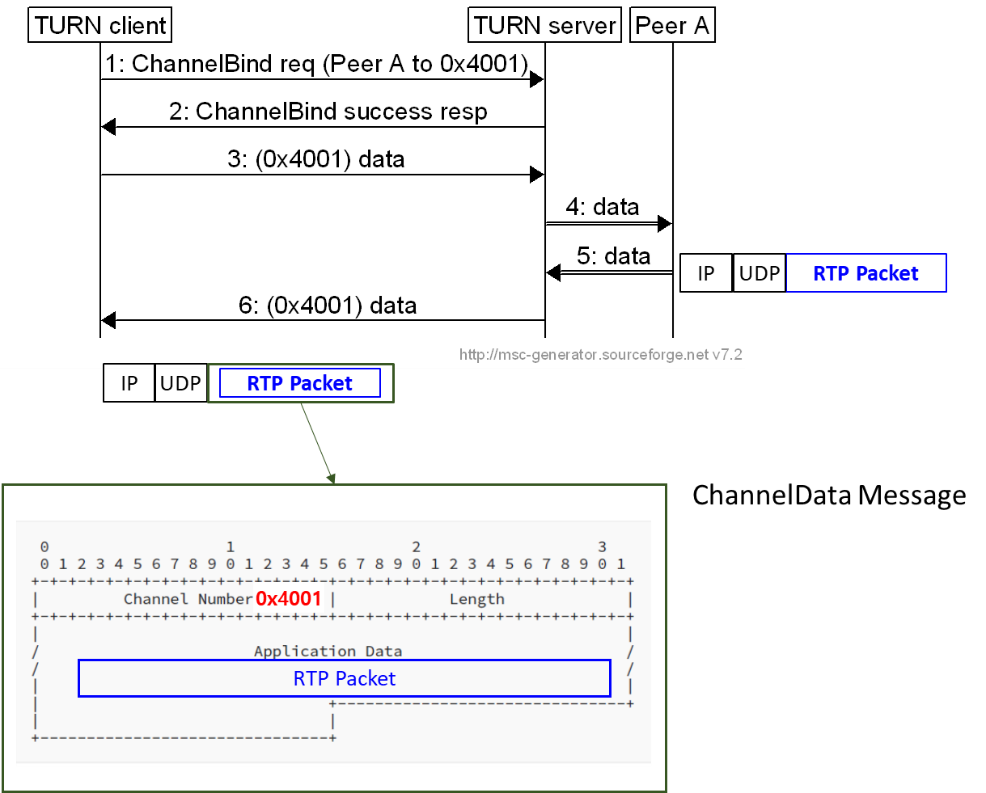
WebRTC uses ICE (Interactive Connectivity Check) [27] to establish connectivity between two endpoints. ICE also makes use of TURN (Traversal Using Relays around NAT) [28] when a direct communication path between two endpoints cannot be found. TURN is an extension to the STUN (Session Traversal Utilities for NAT) [29] which allows a host behind a NAT to request that another host act as a relay. There are two mechanism for the TURN client and its peer to exchange application data using the TURN server. The first mechanism uses the Send and Data methods, the second mechanism uses channels.

The Send and Data mechanism uses Data indications to send application data from the server to the client. In a Data indication, the value portion of the Data attribute contains the application data (e.g., a RTP packet sent from Peer A) as shown in the Figure 11.1-1.

****

**Figure 11.1-1. TURN data delivery using Send/Data indications**

The second mechanism uses an alternate packet format known as the "ChannelData message". The ChannelData message does not use the STUN header used by other TURN messages, but instead has a 4-byte header that includes a number known as a "channel number". Each channel number in use is bound to a specific peer; thus, it serves as a shorthand for the peer's host transport address. If a TURN server receives a UDP datagram from a peer which has a channel number assigned to it, the server encapsulates the data (e.g., a RTP packet) into a ChannelData message and forward it to the client as shown in the Figure 11.1-2.

****

**Figure 11-1-2. TURN data delivery using ChanelData Messages**

**11.2 A typical deployment scenario**

As discussed in the previous section, a TURN server encapsulate the RTP packet sent from a peer into a STUN message or a ChannelData message and then forward it to a TURN client. If the RTP packet contains an RTP HE for PDU Set Marking, UPF needs to identity the delivery mechanism used by a TURN server and extract the RTP packet from Data Indication or ChannelData message containing it.

Figure 11.2-1 shows a deployment example. In this figure UE (TURN client) and TURN server is separated by a NAT. The UE sends TURN messages from its host transport address (198.51.100.2:49721) to the TURN server transport address (192.0.2.15:3478). The UE can learn the TURN server transport address from RTC AF though RTC-5 interface.

The UE uses TURN commends to create and manipulate an ALLOCATION on the server. An allocation is a data structure which consists of

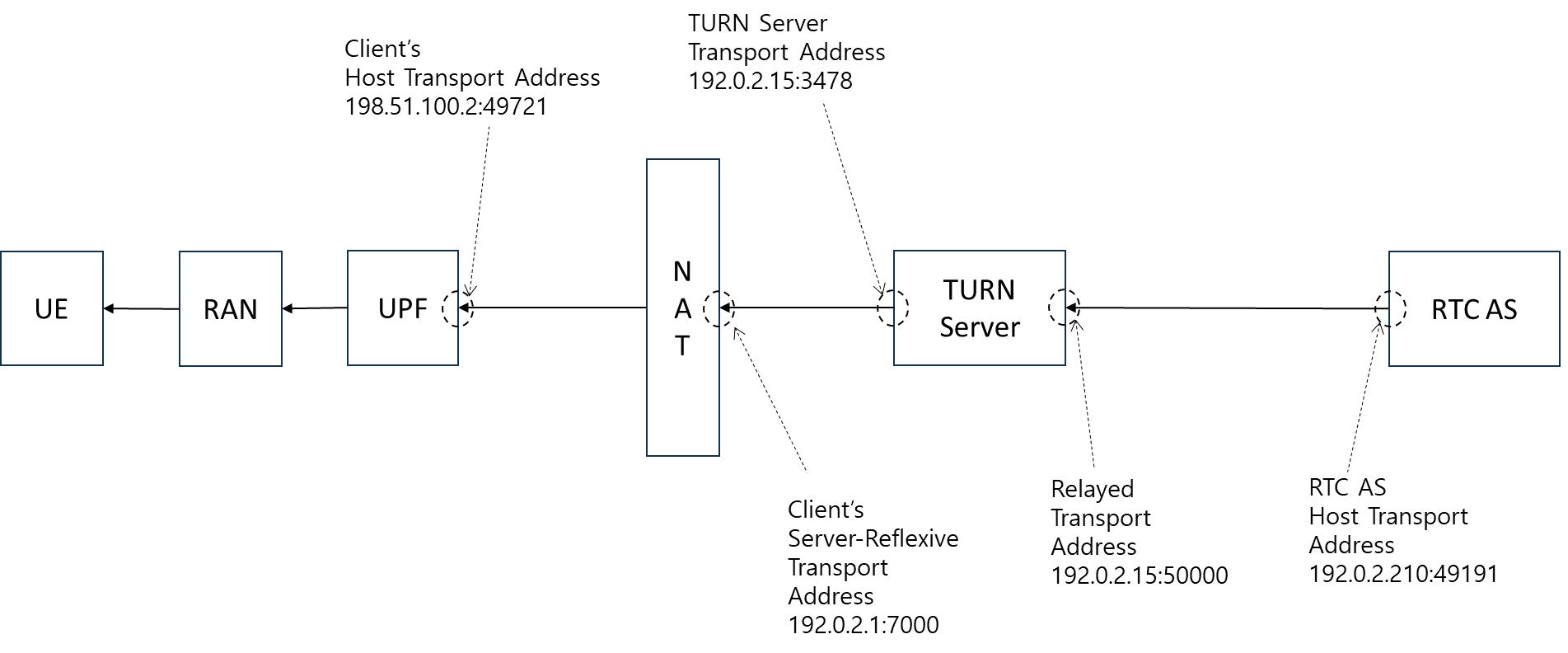
* the relayed transport address or addresses (for IPv4/IPv6 dual stack)
* the 5-tuple: (client’s IP address & port, server IP address & port, transport protocol)
* a list of permissions for each relayed transport address
* a list of channel-to-peer binding for each relayed transport address
* the authentication information and the time-to-expiry for each relayed transport address

The relayed transport address is the transport address allocated by the server for communicating with RTC AS, while the 5-tuple describes the communication path between the client and the server. On the client, the 5-tuple uses the client's host transport address; on the server, the 5-tuple uses the client's server-reflexive transport address.

Each permission consists of an IP address and a lifetime. If the source IP address of the received UDP datagram matches a permission, the TURN server relays the application data to the client; otherwise, the TURN server discards the UDP datagram silently.

In the example in Figure 11.2-1, a downlink traffic from the TURN server to the UE can be described by a 5-tuple ({scrIP, srcPort} = TURN server transport address, {dstIP, dstPort} = Host transport address, protocol = UDP). The relayed transport address can be used to uniquely identify the allocation, we can assume that there is a one-to-one correspondence between a DL service data flow and an allocation (i.e., relayed transport address).

On the other hands, the SDP for a WebRTC service describes the communication session between the TURN server and the RTC AS. Under the assumption that a single instance of RTC AS is involved in the WebRTC service, there is a single media transport between the TURN Server and the RTC AS. It means that there is also a one-to-one correspondence between a DL service data flow and a media transport. As the semantics of multiple "m=" lines using the same transport address are undefined, we can assume that the SDP for the WebRTC service contains a single media description or bundled media descriptions. In both cases, a single local ID value is used for identifying PDU Set Marking RTP HE in the media transport, then that local ID value can identify PDU Set Marking RTP HE in the DL service data flow.



**Figure 11.2-1. A deployment example**

**11.3 Potential solutions**

Note that potential solutions in this clause assume that a single instance of RTC AS is involved in the WebRTC service as discussed in the previous section, i.e., there is a single media transport associated to an allocation.

**11.3.1 PDU Set Marking signaling for Dynamic Policy API**

The DynamicPolicy resource for RTC can contain PDUSetMarking object to describe PDU Set Marking applied to a specific service data flow. Compared to the case when the service data flow carries RTP stream(s) in UDP datagrams, the following properties can be added to the PDUSetMarking object to support PDU Set Marking when a TURN server is used:

* identification of the application layer protocol: RTP or TURN/RTP
* format of the message sent from a TURN server: Data Indication or ChannelData message

**11.3.2 Extension of STUN attributes for PDU Set Marking**

A trusted TURN server can extract the PDU Marking RTP HE form the received UDP datagrams and add it to a Data indication as a dedicated STUN attribute. In order to support this scenario, we can consider the following extension to STUN attributes:

* 3GPP-PDU-SET-MARKING: It can be included in a Data indication to provide PDU Set Marking information. The value of this attributes is same as the value of PDU Set Marking HE in the DATA attribute of the Data indication.
* 3GPP-PDU-SET-INFO: It can be included in an allocation request to provide the local ID value of the PDU Set Marking RTP HE.

**Annex**

**A.1 Connection models to be used in iRTCW**

It is assumed that the services to be supported by iRTCW are real time, with multiple parties, and with some media handling (e.g., VR conferencing, a live performance with a large audience, remote operation supported by multiple experts). The number of UEs can be from two to a large number, e.g., several hundreds. The number of UEs may change during the service, e.g., from two to ten, and to several hundreds. The discussion is what is the best connection model to meet this assumption.

There should be at least four connection models to be considered.

Connection model #1: Multiple UEs are connected via a media server.

Connection model #2: Two UEs are connected via a media server.

Connection model #3: Multiple UEs are connected directly (not via any media servers).

Connection model #4: Two UEs are connected directly (not via any media servers).

It should be noted that Connection models #2 and #4 are introduced for the sake of argument. Connection model #2 can be seen as a subset of Connection model #1. Connection model #4 can be a subset of Connection model #3.

Regarding the necessary network functions, Connection models #1 and #2 need a media server as an extra entry, while Connection models #3 and #4 do not.

When it comes to a conference service, the media server in Connection models #1 and #2 can serve as a conference room, which can be seen by a UE as an intermediate target destination (as a rendezvous point) instead of the real peer UEs. The conference room provided by the media server allows a calling UE to stay there, does not require immediate discovery of the real target UEs, and thus does not require the registration of the peer UEs. In Connection models #3 and #4, the conference session should be set up among all UEs by exchanging information (by SDP offer/answer) among them. Exchanging the information, which is performed by control plane signalling, needs to resolve the exact destination of the peer UEs, which should be registered in advance. In other words, Connection models #3 and #4 require a network entity for UE registration.

The iRTC service may need connections between a wide range of the numbers of UEs (from two parties to multiple parties, even scalable to hundreds of parties). Considering the scalability, a large number of direct connections between multiple UEs (by Connection model #3) have a severe issue for network performance. The number of media paths increases by the square of the number of UEs, which easily leads to network congestion.

The number of the parities may change during a call/service. Direct connection model (by Connection model #4) may be best for a two-party call, but how to support additional UEs during a call should be considered. Connection model #4 can easily transit to Connection model #3, which leads to the network congestion issue. If Connection model #4 transits to Connection model #1, it involves a complex process for transition. For supporting modification of a call by increasing and decreasing the number of UEs, using a media server (by Connection models #1 or #2) is simple and scalable.

Regarding the communication delay, connections through a media server have a certain additional delay and direct connections have a shorter delay.

In some services, such as conferencing [Conferencing-oriented], the basic connection model is #1 or #2 (connection via media server). In other services and purposes, such as split rendering, the basic connection model is #3 or #4 (direct connection between endpoints).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Multiple UEs connected via a media server  (Connection model #1) | Two UEs connected via a media Server (Connection model #2) | Multiple UEs connected directly (Connection model #3) | Two UEs connected directly (Connection model #4) |
| NW/UE functions specific to the connection model | Media server | | - UE registration to receive incoming calls  - Direct reachability between UEs (i.e., IPv6 global unicast address or IPv4 and Full-cone NAT) | |
| Responsiveness of call modification (increase/decrease the number of UEs) | High (UEs can join and leave the call while maintaining the same connection model) | | Low (especially for increasing the number of UEs, full-mesh connection has severe network performance issue for scalability) | Low (when additional UEs joins the call, the connection model may move to Connection model #1 or #3) |
| Communication Delay | Ordinary RTT using intermediate server for conferencing | | Shorter delay than with a media server | |



Connection model #1



Connection model #2



Connection model #3



Connection model #4

**A.2 WebRTC signalling protocol**

**A.2.1 General**

The Simple WebRTC Application Protocol (SWAP) supports collaboration scenario 3 described in 3GPP TS 26.506 [x5].

NOTE: The signalling protocol which supports collaboration scenario 4 (and applicable to collaboration scenario 3) is specified as different protocol in future release.

**A.2.2 Protocol and version identification**

The WebRTC signalling protocol and the version of the protocol shall be determined per WebSocket connection. The WebRTC signalling protocol and the version of the protocol shall be identified by the WebSocket URI for the HTTP upgrade request for WebSocket connection establishment (i.e., the Request-URI of the HTTP request). The WebSocket URI for the HTTP upgrade request shall be consistent with the WebSocket URI structure specified in clause A.2.3.

The use of "Sec-WebSocket-Protocol" header field is dependent on the WebRTC signalling protocol and the version of the protocol.

**A.2.3 WebSocket URI structure**

WebSocket URI of WebSocket connection for WebRTC signalling protocol message shall be:

{protocolRoot}/<protocolName>/<protocolVersion>

"protocolRoot" shall be a concatenation of the following parts:

* scheme ("wss")
* the fixed string "://"
* authority (host and optional port) as defined in IETF RFC 3986. The host should be represented by the service provider (operator or OTT) specific FQDN (for FQDN examples see clause 28.3.2 in 3GPP TS 23.003).
* an optional deployment-specific string (e.g., server prefix) that starts with a "/" character.

"protocolName" shall be protocol-specific string which indicates the name of the WebRTC signalling protocol.

"protocolVersion" shall indicate the version of the WebRTC signalling protocol. The protocol version shall be indicated as the concatenation of the letter "v" and the WebRTC signalling protocol version number. The other fields shall not be included in the URI.

For example, 'v1'.

NOTE: The "protocolVersion" will only be increased if the new protocol version contains not backward compatible changes.

A URI should not contain a trailing slash, and if it contains one, then it should be ignored/removed.

**A.2.4 SWAP**

**A.2.4.1 Protocol and version identification**

The SWAP version shall be included in the WebSocket URI path as “/3gpp-swap/v1/".

The present version of SWAP, the Sec-WebSocket-Protocol header field with "3gpp.SWAP.v1" subprotocol identifier shall be included in the HTTP upgrade request.

**A.2.4.2 Transport**

SWAP protocol shall operate over a full-duplex reliable WebSocket connection between the two endpoints or between an endpoint and a SWAP server. The following figure depicts both scenarios.



In the former, one of the endpoints shall act as the WebSocket server and listen for the incoming connection request. The endpoint is not required to support more than one client connection at any point of time.

When a SWAP server is used, sufficient information shall be provided to facilitate the relaying of the messages from the server to the other endpoint.

**A.2.4.3 State machine**

The SWAP server maintains state information about ongoing WebRTC sessions. The following state machine reflects the state tracked by the SWAP server.



The SWAP protocol is designed to adhere to the JSON Session Establishment Protocol (JSEP) state machine as defined in RFC8829. The JSEP state machine is reproduced in the following figure.

Diagram

Description automatically generated

SWAP currently does not support preliminary answers in its version 1. Any preliminary answers that are generated by the application will not be sent by the SWAP endpoint.

SWAP version 1 does not support ICE trickling. The final list of ICE candidates is expected to be part of the initial offer message. The application shall wait for the ICE gathering phase to finish prior to sending the offer to the remote endpoint.

**A.2.4.4 Message syntax and semantics**

**A.2.4.4.1 Common message fields**

**A.2.4.4.1.1 Source Id**

Each message shall carry a unique source identifier that identifies the message source. The source identifier shall be a randomly generated string. The source identifier shall not be changed during the lifetime of a session.

A SWAP server that detects a change in the source identifier from an endpoint over the same WebSocket connection shall ignore the corresponding message. The source identifier shall at least have 10 UTF-8 characters.

**A.2.4.4.1.2 Message Id**

The message identifier shall be a sequence number for the message. The message identifier is scoped by the source identifier, i.e. it shall be uniquely assigned by the source of the message.

The message identifier shall be a positive monotonically increasing number.

Message Type

The message type identifies the type of the SWAP message. The supported message types in version 1 of the specification are:

* Register
* Response
* Connect
* Accept
* Reject
* Update
* Close
* Application.

**A.2.4.4.2 Register message**

An endpoint registers with the SWAP server and provides the matching criteria that may be used to match this endpoint with incoming connection requests.

The register message is not required for the case of a direct connection between the two endpoints.

**A.2.4.4.2.1 Parameters**

matching\_criteria: an object that provides the matching criteria for relaying incoming SWAP messages to their destination. The matching criteria object consists of a type and a value.

The supported types in this version of the specification are the following:

* ipv4: The IPv4 address of the target endpoint
* ipv6: The IPv6 address of the target endpoint
* fqdn: The FQDN of the target endpoint
* service: An identifier of a service or an application
* user: An identifier of the user such as a SIP address, a GPSI, or an MSISDN
* eas: An EAS identifier
* app: application-specific matching criteria that is compared using binary or string comparison
* location: one or more identifiers of a geographic location or area
* qos: a description of the QoS that is supported by the connection to the endpoint
* processing: a profile description of the processing capabilities of the endpoint.

The matching criteria may be combined together to further restrict the selection of the target endpoint. If multiple endpoints match all provided criteria, then the SWAP server shall randomly select one of the target endpoints.

An endpoint that registers without providing certain matching criteria, such as qos or processing, shall be deprioritized during the selection process, where the request contain these matching criteria.

**A.2.4.4.3 Response message**

A SWAP server shall respond to every received request with a response message. The response message shall indicate whether the message is acknowledged or erroneous.

If a message is relayed properly to an endpoint, an acknowledgement message shall be sent to the source endpoint.

If an error is detected or a target endpoint cannot be identified, the SWAP server shall respond with an error response to the source endpoint.

In addition to the common fields, the response message shall include the request message id. In case of an error response, the message shall contain a textual description of the error.

**A.2.4.4.3.1 Parameters**

type: the type parameter may either be “ack” or “error”

source: the source identifier of the message source

request: the message identifier of the request

description: a description of the error message.

**A.2.4.4.4 Connect message**

The connect message is used by the source to establish a connection with the endpoint. The request shall include the SDP offer. If connecting via a SWAP server, the request shall include the matching\_criteria parameter to identify the target endpoint.

**A.2.4.4.4.1 Parameters**

offer: a string that includes the SDP description for the offer.

matching\_criteria: an array that contains the matching criteria for the target endpoint. Each object shall be comply with the definition of a matching criteria as described in clause 5.4.2.4.2.1.

**A.2.4.4.5 Accept message**

If the connection request is accepted by the remote endpoint, it shall reply with an accept message. The accept message shall contain the answer SDP.

**A.2.4.4.5.1 Parameters**

answer: This parameter shall contain the answer SDP.

**A.2.4.4.6 Update message**

The update message may be sent by any of the endpoints of a WebRTC session. It contains the updated SDP, which may add, update, or remove one or more local media streams. If accepted, the remote endpoint shall reply with an accept message.

**A.2.4.4.6.1 Parameters**

sdp: The updated local SDP that is transmitted to the remote endpoint.

**A.2.4.4.7 Reject message**

In case the remote endpoint does not accept the offer or update message, it shall respond with the reject message. The message shall contain a reference to the corresponding offer or update message as well as a description of the reason why the message was rejected.

**A.2.4.4.7.1 Parameters**

source: the source identifier of the message source

request: the message identifier of the request

error\_id: an identifier of the error message

description: a description of the error message.

**A.2.4.4.8 Close message**

The close message may be triggered by any of the two endpoints of a WebRTC session. Upon reception, the endpoint shall respond with an accept message, after which the WebRTC session is torn down and the resources associated with the WebRTC session are released.

**A.2.4.4.9 Application message**

Application-specific message may be defined by the application and exchanged between the endpoints of a WebRTC session. The message shall contain a type that uniquely identifies the type of the application message. If an application message type is not supported, it shall be rejected by the remote endpoint.

**A.2.4.4.9.1 Parameters**

type: the type of the application message shall be a URN that uniquely identifies the application message type.

value: an object that contains the application message content.

**A.2.4.5 Integrity and security**

Integrity and confidentiality protection are supported through the protection of the message information as follows:

* A key derivation mechanism is configured by the application provider to the session participants, e.g. using a shared secret algorithm
* For integrity protection, the derived key is used to provide integrity protection, e.g. using a Message Authentication Code (MAC) for message payload
* For encryption, the derived key is used to encrypt the message payload. The encrypted data may then be encoded using base64 to enable embedding it in JSON.

These mechanisms are possible to implement using the WebCrypto API, which makes them web-friendly. Consulting with SA3 on these security algorithms is recommended.

**A.2.4.6 JSON schema**

The JSON schema of the SWAP messages is follows:

{

"$schema": "http://json-schema.org/draft-07/schema",

"title": "3GPP.SWAP",

"type": "object",

"description": "The description of the SWAP messages",

"properties": {

"version": {

"description": "the version of the SWAP protocol",

"type": "integer"

},

"source\_id": {

"description": "A unique identifier of the source",

"type": "string"

},

"message\_id": {

"description": "the sequence number of the message ",

"type": "integer"

},

"message\_type": {

"description": "the type of the SWAP message",

"type": "string",

"enum": ["register", "connect", "response", "accept", "reject", "update", "close", "application"]

},

"oneOf": [

{

"type": "object",

"properties": {

"matching\_criteria": {"type": "string", "enum": ["ipv4", "ipv6", "fqdn", "service", "user", "eas", "app", "location", "qos", "processing"]}

}

},

{

"type": "object",

"properties": {

"type": {"type": "string", "enum": ["ack", "error"]},

"source": {"type": "string"},

"request": {"type": "integer"},

"description": {"type": "string"}

}

},

{

"type": "object",

"properties": {

"offer": {"type": "string"},

"matching\_criteria": {"type": "string", "enum": ["ipv4", "ipv6", "fqdn", "service", "user", "eas", "app", "location", "qos", "processing"]}

}

},

{

"type": "object",

"properties": {

"answer": {"type": "string"}

}

},

{

"type": "object",

"properties": {

"source": {"type": "string"},

"request": {"type": "number"},

"error\_id": {"type": "string"},

"description": {"type": "string"}

}

},

{

"type": "object",

"properties": {

"type": {"type": "string"},

"value": {"type": "object"}

}

}

],

"extensions": {}

},

"required": ["version", "source", "message\_id"]

}

**References**

[1] [Azure Kinect DK hardware specifications, Microsoft](https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification)

[2] [Intel RealSense D400 series product family datasheet, Intel](https://dev.intelrealsense.com/docs/intel-realsense-d400-series-product-family-datasheet)

[3] [Camera calibration parameters, MathWorks](https://www.mathworks.com/help/vision/ug/camera-calibration.html#bu0nh2_)

[4] [intrinsicMatrix, Apple](https://developer.apple.com/documentation/avfoundation/avcameracalibrationdata/2881135-intrinsicmatrix)

[5] [CameraIntrinsics, Google](https://developers.google.com/ar/reference/java/com/google/ar/core/CameraIntrinsics)

[6] [S4aR220046, 3D video types for iRTC client in the terminal, Meta](https://www.3gpp.org/ftp/TSG_SA/WG4_CODEC/3GPP_SA4_AHOC_MTGs/SA4_RTC/Docs/S4aR220046.zip)

[7] [Quick measure, Samsung](https://play.google.com/store/apps/details?id=com.samsung.android.ruler&hl=en_US&gl=US)

[8] [Measure app, Apple](https://support.apple.com/en-us/HT208924#:~:text=The%20Measure%20app%20uses%20augmented,a%20photo%20of%20the%20measurement)

[9] [S4aR220014, 3D video capture description for iRTC client in the terminal, Meta](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/3GPP_SA4_AHOC_MTGs/SA4_RTC/Docs/S4aR220014.zip)

[10] [Windows microphone array geometry descriptor format, Microsoft](https://docs.microsoft.com/en-us/windows-hardware/drivers/audio/microphone-array-geometry-descriptor-format)

[11] VRIF volumetric video guidelines v0.95, <https://www.vr-if.org/guidelines/>

[12] 3GPP TSG SA WG4 [S4-221150](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_120-e/Docs/S4-221150.zip), “MeCAR Permanent Document v0.3.0”, August 2022

[13] 3GPP TSG SA WG4 [S4-221209](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_120-e/Docs/S4-221209.zip), “5G\_RTP Permanent Document v0.0.2”, August 2022

[14] IETF RFC 4566, “SDP: Session Description Protocol”, July 2008

[15] IETF RFC 8831, "WebRTC Data Channels", January 2021

[16] IETF RFC 4960, “Stream Control Transmission Protocol”, September 2007

[17] IETF RFC 8837, “Differentiated Services Code Point (DSCP) Packet Markings for WebRTC QoS"

[18] IETF RFC 8864, “Negotiation Data Channels Using the Session Description Protocol (SDP)”, January 2021

[19] IETF RFC 8832, “WebRTC Data Channel Establishment Protocol”, January 2021

[20] 3GPP TS 26.928, “Extended Reality (XR) in 5G”, April 2022

[21] ISO/IEC [23090-5:2021](https://www.iso.org/standard/73025.html): “Information technology — Coded representation of immersive media — Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-PCC)”

[22] C. Cao, T. Simon, J. Kim, G. Schwartz, M. Zollhoefer, S. Saito, S. Lombardi, S. Wei, D. Belko,

S. Yu, Y. Sheikh, J. Saragih, [Authentic volumetric avatars from a phone scan](https://dl.acm.org/doi/abs/10.1145/3528223.3530143), ACM Transactions on Graphics, Volume 41, Issue 4, July 2022

[23] [Multiface dataset, Meta](https://github.com/facebookresearch/multiface)

[24] S. Ma, T. Simon, J. Saragih, D. Wang, Y. Li, F. De La Torre, and Y. Sheikh, [Pixel codec avatars](https://www.computer.org/csdl/proceedings-article/cvpr/2021/450900a064/1yeMmobLlwQ), IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2021

[25] <https://www.youtube.com/watch?v=t7_TMD7v0Xs>

[26] [S4-230073, An implementation of real-time V3C streaming for conversational scenario](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_122_Athens/Docs/S4-230073.zip)

[27] IETF RFC 8445, “Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal”

[28] IETF RFC 8656, “Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)

[29] IETF RFC 8489, “Session Traversal Utilities for NAT (STUN)”

# **Revision history**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Meeting** | **Subject / Comment** | **Old** | **New** |
| 2022-05-20 | SA4#119-e | - PD skeleton ([S4-220769](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_119-e/Docs/S4-220769.zip)) | 0.10 | 0.11 |
| 2022-11-14 | SA4#121 | - 3D video capture ([S4aR220014](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/3GPP_SA4_AHOC_MTGs/SA4_RTC/Docs/S4aR220014.zip))  - Microphone description ([S4aR220043](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/3GPP_SA4_AHOC_MTGs/SA4_RTC/Docs/S4aR220043.zip))  - 3D size measurement and scaling ([S4aR220023](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/3GPP_SA4_AHOC_MTGs/SA4_RTC/Docs/S4aR220023.zip))  - WebRTC QoS architecture ([S4aR220010](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/3GPP_SA4_AHOC_MTGs/SA4_RTC/Docs/S4aR220010.zip))  - Dynamic 3D representation use cases and requirements ([S4-221193](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_120-e/Docs/S4-221193.zip)) | 0.11 | 0.20 |
| 2023-02-20 | SA4#122 | - Real-time metadata transport over data channel ([S4-221557](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_121_Toulouse/Docs/S4-221557.zip))  - Additions to size measurement of 3D objects in iRTCW ([S4-221546](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_121_Toulouse/Docs/S4-221546.zip))  - iRTCW architecture for AR conferencing ([S4-221547](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_121_Toulouse/Docs/S4-221547.zip))  - Proposal for connection models to be used in iRTCW ([S4-221549](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_121_Toulouse/Docs/S4-221549.zip))  - 3D avatar generation & operation for iRTC client in the terminal ([S4aR230011](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/3GPP_SA4_AHOC_MTGs/SA4_RTC/Docs/S4aR230011.zip))  - XR streaming use case ([S4-230389](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_122_Athens/Docs/S4-230389.zip))  - APIs for AR conferencing ([S4-230319](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_122_Athens/Docs/S4-230319.zip))  - Simple WebRTC Application Protocol (SWAP) ([S4-230344](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_122_Athens/Docs/S4-230344.zip))  - AR call solution for smartphones or tablets ([S4aR230060](https://www.3gpp.org/ftp/TSG_SA/WG4_CODEC/3GPP_SA4_AHOC_MTGs/SA4_RTC/Docs/S4aR230060.zip)) | 0.20 | 0.30 |
| 2023-04-20 | SA4#123-e | - V3C pipeline for iRTC ([S4-230574](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_123-e/Docs/S4-230574.zip)) | 0.30 | 0.50 |
| 2023-11-16 | SA4#126 | - ProvisionedConfiguration and PolicyTemplate resources ([S4-231960](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_126_Chicago/Docs/S4-231960.zip))  - MetricsReportingConfiguration resource ([S4-231898](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_126_Chicago/Docs/S4-231898.zip))  - PDU Set Marking with TURN ([S4-231782](https://www.3gpp.org/ftp/tsg_sa/WG4_CODEC/TSGS4_126_Chicago/Docs/S4-231782.zip)) | 0.5 | 0.6 |