

声网

Web媒体处理与实时传输标准实践

Practices of Web Media Processing and Real-time Communication Standards

高纯

Outline

1. Background
2. Case1: E2E Encryption
3. Case2: Digital Rights Management
4. Case3: H265 Supporting for RTC
5. Case4: Alpha Video Transmission

Background

New Trends in the RTC Industry



Expanding overseas business



Web is the most important platform overseas



Security and compliance by design is required by foreign laws

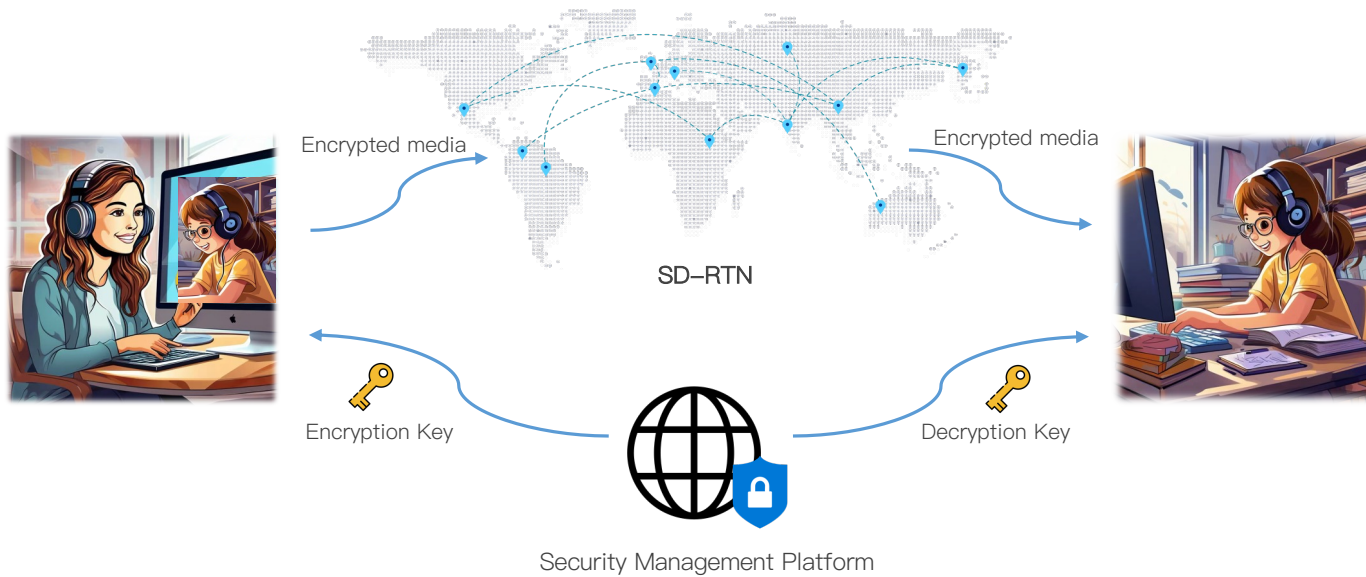


Diverse application scenarios create diverse requirements on web infrastructure

Case 1: E2E Encryption

Providing reliable transmission network

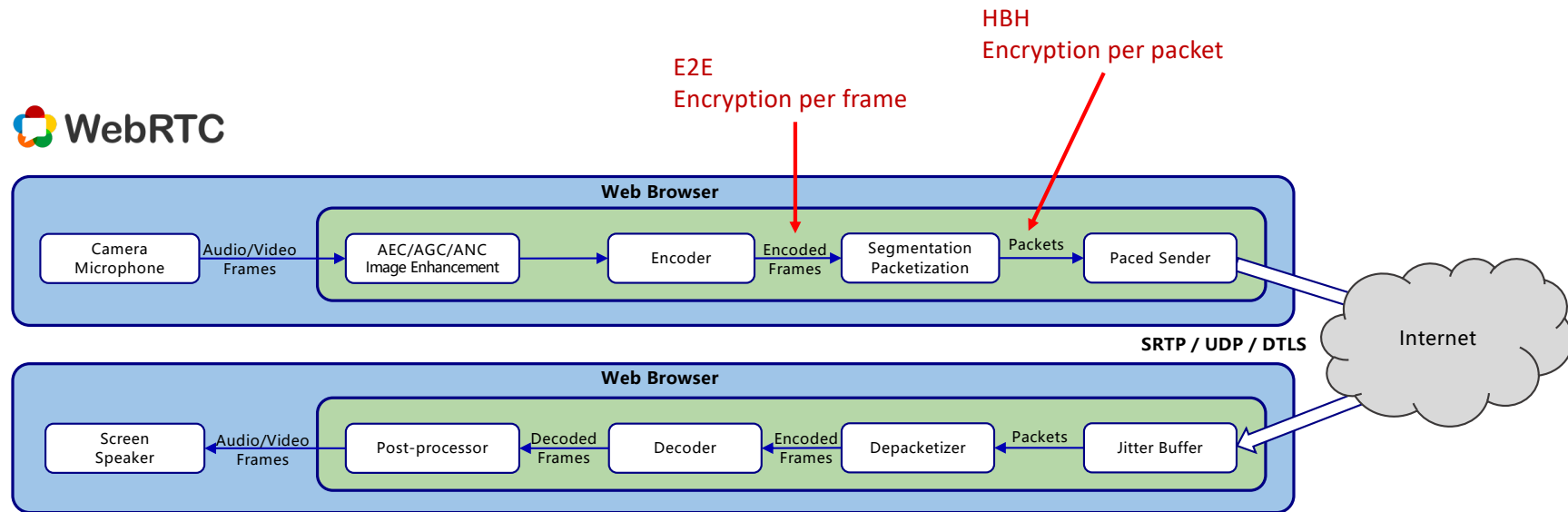
- Encrypt and decrypt media by end device to protect privacy
- User data could not be decrypted by RTC service provider



Case 1: E2E Encryption

Requirements:

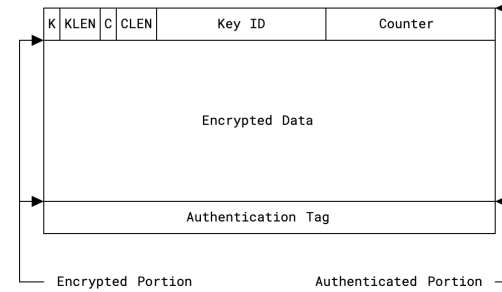
- Provide interface to grab encoded media data
- Provide high performance encryption/decryption component
- Provide interface to write back transformed media data



Case 1: E2E Encryption

Secure Frame (SFrame)

- End-to-end encryption and authentication mechanism for media frames
- Compatible with RTP & non-RTP media transport
- Reduce bandwidth overhead by adding encryption overhead only once per media frame, instead of once per packet.



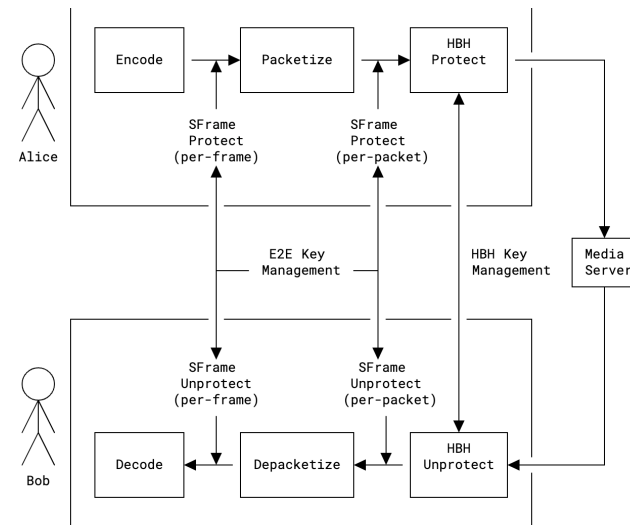
Workgroup: Network Working Group
 Internet-Draft: draft-ietf-sframe-enc-04
 Published: 22 October 2023
 Intended Status: Standards Track
 Expires: 24 April 2024
 Authors: E. Omara (Apple), J. Uberti (Google), S. Murillo (CoSMo Software), R. L. Barnes, Ed. (Cisco), Y. Fablet (Apple)

Secure Frame (SFrame)

Abstract

This document describes the Secure Frame (SFrame) end-to-end encryption and authentication mechanism for media frames in a multiparty conference call, in which central media servers (selective forwarding units or SFUs) can access the media metadata needed to make forwarding decisions without having access to the actual media.

The proposed mechanism differs from the Secure Real-Time Protocol (SRTP) in that it is independent of RTP (thus compatible with non-RTP media transport) and can be applied to whole media frames in order to be more bandwidth efficient.



Case 1: E2E Encryption

SFrameTransform

Spec: <https://www.w3.org/TR/webrtc-encoded-transform/>

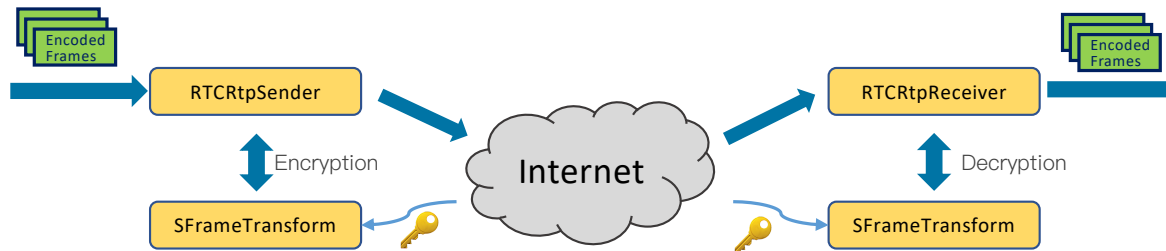
```
typedef (SFrameTransform or RTCRtpScriptTransform) RTCRtpTransform;

// New methods for RTCRtpSender and RTCRtpReceiver
partial interface RTCRtpSender {
    attribute RTCRtpTransform? transform;
};
partial interface RTCRtpReceiver {
    attribute RTCRtpTransform? transform;
};
```

```
enum SFrameTransformRole {
    "encrypt",
    "decrypt"
};
dictionary SFrameTransformOptions {
    SFrameTransformRole role = "encrypt";
};

typedef [EnforceRange] unsigned long long SmallCryptoKeyID;
typedef (SmallCryptoKeyID or bigint) CryptoKeyID;

[Exposed=(Window,DedicatedWorker)]
interface SFrameTransform : EventTarget {
    constructor(optional SFrameTransformOptions options = {});
    Promise<undefined> setEncryptionKey(CryptoKey key, optional CryptoKeyID keyID);
    attribute EventHandler onerror;
};
```



Case 1: E2E Encryption

web-platform-tests dashboard

Sign in with GitHub

Latest Run Recent Runs Interop 2024 Insights Processor About

wpt / webrtc-encoded-transform

partly

For information on the search syntax, [view the search documentation](#)

Showing 5 tests (11 subtests) in webrtc-encoded-transform from the latest master test runs for chrome[experimental], edge[experimental], firefox[experimental], safari[experimental]

LINK EDIT

| Path | Chrome 126 Linux 20.04 53eba69 May 16, 2024 | Edge 126 Windows 10.0 53eba69 May 16, 2024 | Firefox 128 Linux 20.04 53eba69 May 16, 2024 | Safari 194 preview macOS 13.6 53eba69 May 16, 2024 |
|---------------------------------------|--|---|---|---|
| ^ | ^ | ^ | ^ | ^ |
| sframe-keys.https.html | 0 / 2 | 0 / 2 | 0 / 2 | 0 / 2 |
| sframe-transform-buffer-source.html | 0 / 1 | 0 / 1 | 0 / 1 | 0 / 1 |
| sframe-transform-in-worker.https.html | 0 / 1 | 0 / 1 | 0 / 1 | 0 / 1 |
| sframe-transform-readable.html | 0 / 1 | 0 / 1 | 0 / 1 | 0 / 1 |
| sframe-transform.html | 0 / 6 | 0 / 6 | 0 / 6 | 0 / 6 |
| Subtest Total | 0 / 11 | 0 / 11 | 0 / 11 | 0 / 11 |

Case 1: E2E Encryption

Partly testable on Safari with feature flag enabled

Functionalities not ready

The image shows two overlapping browser windows. The top window is Safari's 'Feature Flags' page, with 'WebRTC SFrame Transform API' checked and labeled as 'Testable'. The bottom window is the WebKit Bugzilla page for bug 218752, 'Add a WebRTC SFrame transform'. The bug status is 'RESOLVED FIXED', reported on 2020-11-10, and modified on 2020-11-16. The component is 'WebKit' and the assignee is 'youenn fablet'.

The image shows a Jest test runner output in a browser window. It displays a summary of 6 tests, with 3 passing and 3 failing. The 'Details' section shows a table of test results:

| Result | Test Name | Message |
|--------|---|---|
| PASS | Cannot reuse attached transforms | Asserts run: PASS assert_throws_dom("InvalidStateError", function () => send...) |
| PASS | SFrameTransform exposes readable and writable | Asserts run: PASS assert_true(true) #webrtc-encoded-transform/#sframe-transform.html:35:16 |
| PASS | readable/writable are locked when attached and after being attached | Asserts run: PASS assert_true(true) #webrtc-encoded-transform/#sframe-transform.html:36:16 |
| FAIL | SFrame with array buffer - authentication size 10 | promise_test: Unhandled rejection with value: object "TypeError: undefined is not an object (evaluating 'crypto.subtle.importKey')" |
| FAIL | SFrame decryption with array buffer that is too small | promise_test: Unhandled rejection with value: object "TypeError: undefined is not an object (evaluating 'crypto.subtle.importKey')" |
| FAIL | SFrame transform gets errored if trying to process unexpected value types | promise_test: Unhandled rejection with value: object "TypeError: undefined is not an object (evaluating 'crypto.subtle.importKey')" |

Case 1: E2E Encryption

RTCRtpScriptTransform

Spec: <https://www.w3.org/TR/webrtc-encoded-transform/>

```
typedef (SFrameTransform or RTCRtpScriptTransform) RTCRtpTransform;

// New methods for RTCRtpSender and RTCRtpReceiver
partial interface RTCRtpSender {
  attribute RTCRtpTransform? transform;
};
partial interface RTCRtpReceiver {
  attribute RTCRtpTransform? transform;
};
```

```
partial interface RTCRtpSender {
  Promise<undefined> generateKeyFrame(optional sequence <DOMString> rids);
};
```

```
[Exposed=DedicatedWorker]
interface RTCTransformEvent : Event {
  readonly attribute RTCRtpScriptTransformer transformer;
};

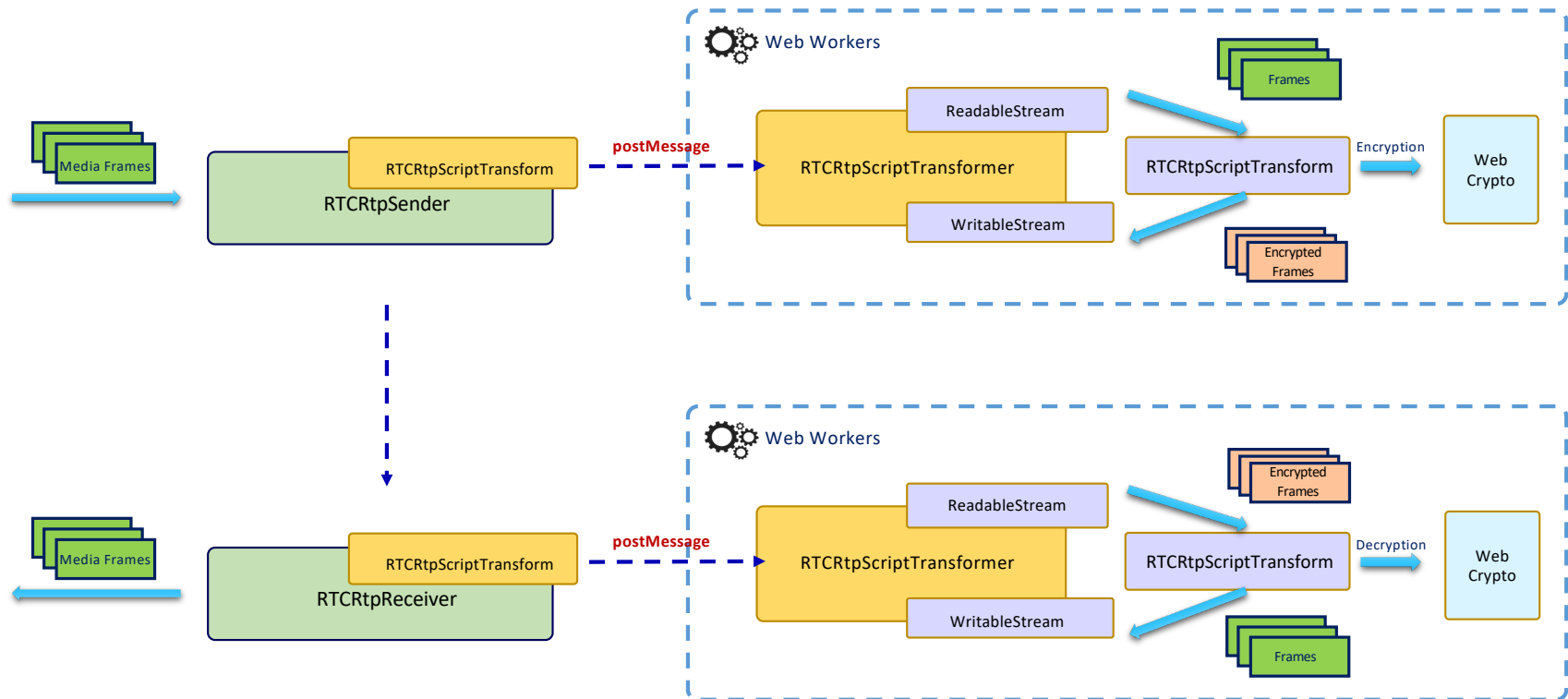
partial interface DedicatedWorkerGlobalScope {
  attribute EventHandler onrtctransform;
};

[Exposed=DedicatedWorker]
interface RTCRtpScriptTransformer : EventTarget {
  // Attributes and methods related to the transformer source
  readonly attribute ReadableStream readable;
  Promise<unsigned long long> generateKeyFrame(optional DOMString rid);
  Promise<undefined> sendKeyFrameRequest();
  // Attributes and methods related to the transformer sink
  readonly attribute WritableStream writable;
  attribute EventHandler onkeyframerequest;
  // Attributes for configuring the Javascript code
  readonly attribute any options;
};

[Exposed=Window]
interface RTCRtpScriptTransform {
  constructor(Worker worker, optional any options, optional sequence<object> transfer);
};
```

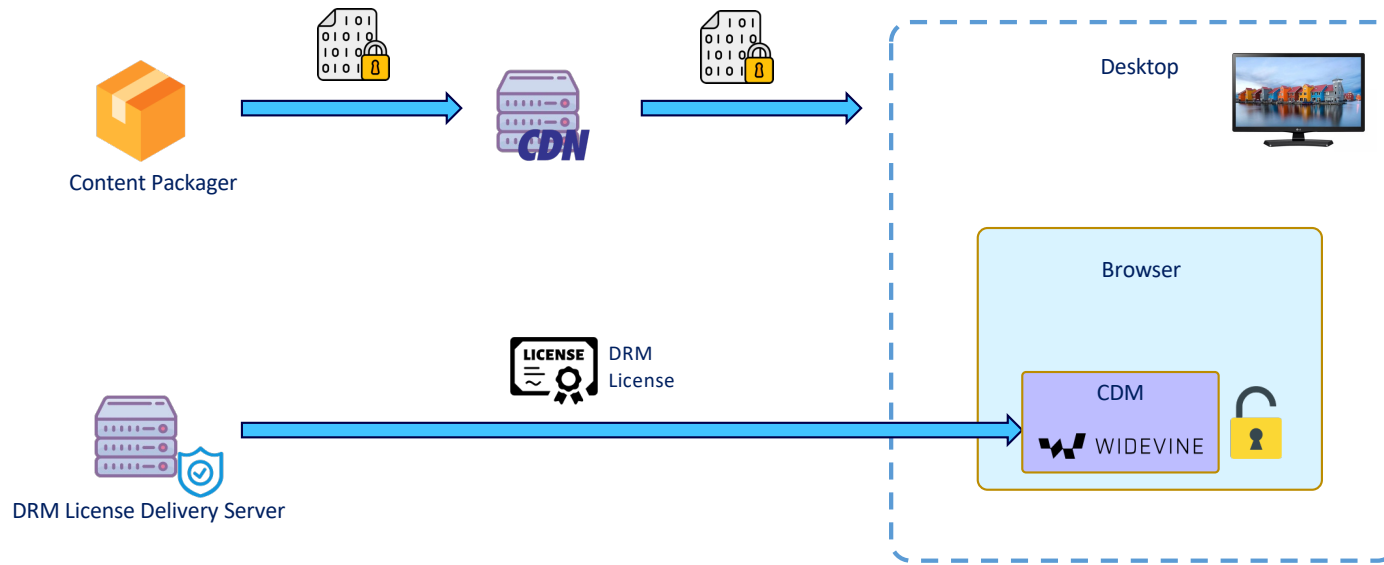
Case 1: E2E Encryption

E2E encryption with RTCRtpScriptTransform and Web Crypto API



Case 2: Digital Rights Management

Traditional DRM for CDN one-way media



Case 2: Digital Rights Management

WebRTC Extended Use Cases

Case: Live encoded non-WebRTC media

| Requirement ID | Description |
|----------------|--|
| N40 | An application can create an outgoing WebRTC connection without activating an encoder. |
| N41 | An application can create encoded video frames from encoded data and metadata, and enqueue them on an outgoing WebRTC connection |
| N42 | The WebRTC connection can generate signals indicating the desired bandwidth, and surface those to the application. |

Case: Transmitting stored encoded media

| Requirement ID | Description |
|----------------|---|
| N41 | An application can create encoded video frames from encoded data and metadata, and enqueue them on an outgoing WebRTC connection |
| N42 | The WebRTC connection can generate signals indicating the desired bandwidth, and surface those to the application. |
| N43 | The application can modify metadata on outgoing frames so that they fit smoothly within the expected sequence of timestamps and sequence numbers. |
| N44 | The application can signal the WebRTC encoder when resuming live transmission in such a way that generated frames fit smoothly within the expected sequence of timestamps and sequence numbers. |

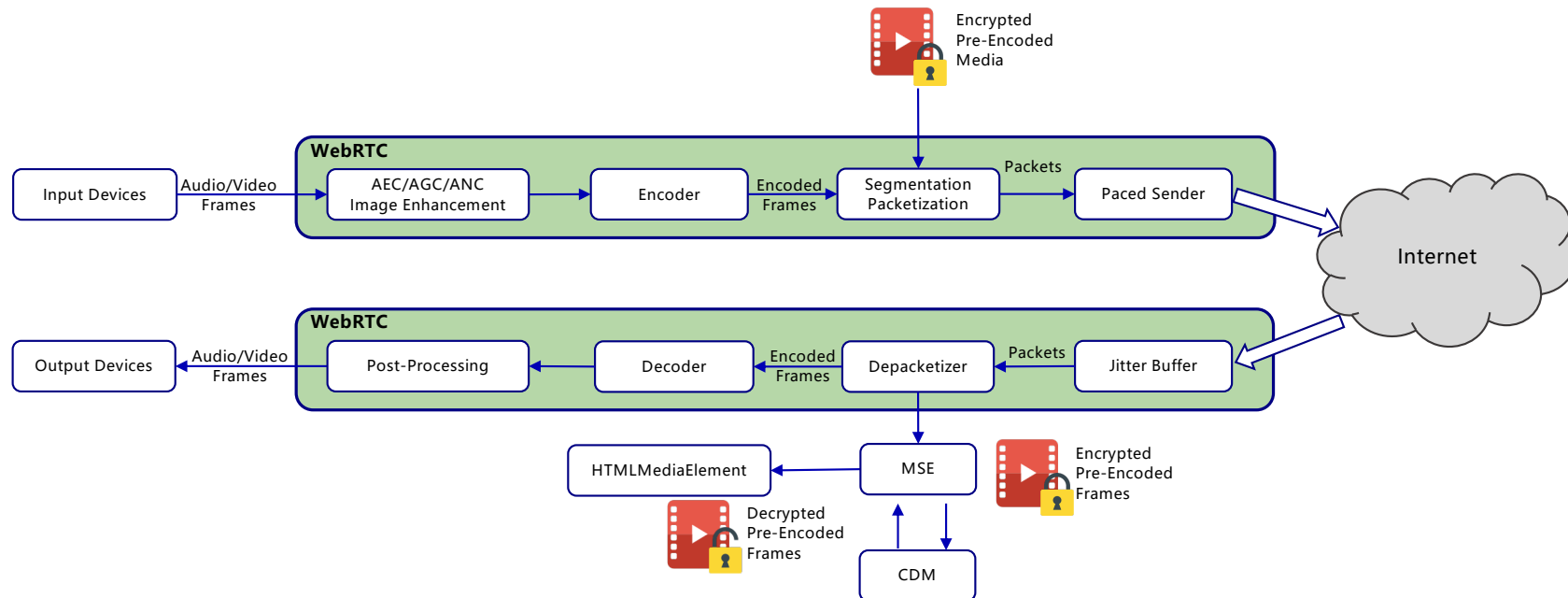
Case: Decoding pre-encoded media

| Requirement ID | Description |
|----------------|---|
| N45 | An application can create an incoming WebRTC connection to accept frames as if they were coming in over RTP, without creating an RTP transport. |
| N46 | An application can create encoded video frames from encoded data and metadata, and enqueue them on an incoming WebRTC connection. |
| N47 | The WebRTC connection can generate signals indicating demands for keyframes, and surface those to the application. |

Case 2: Digital Rights Management

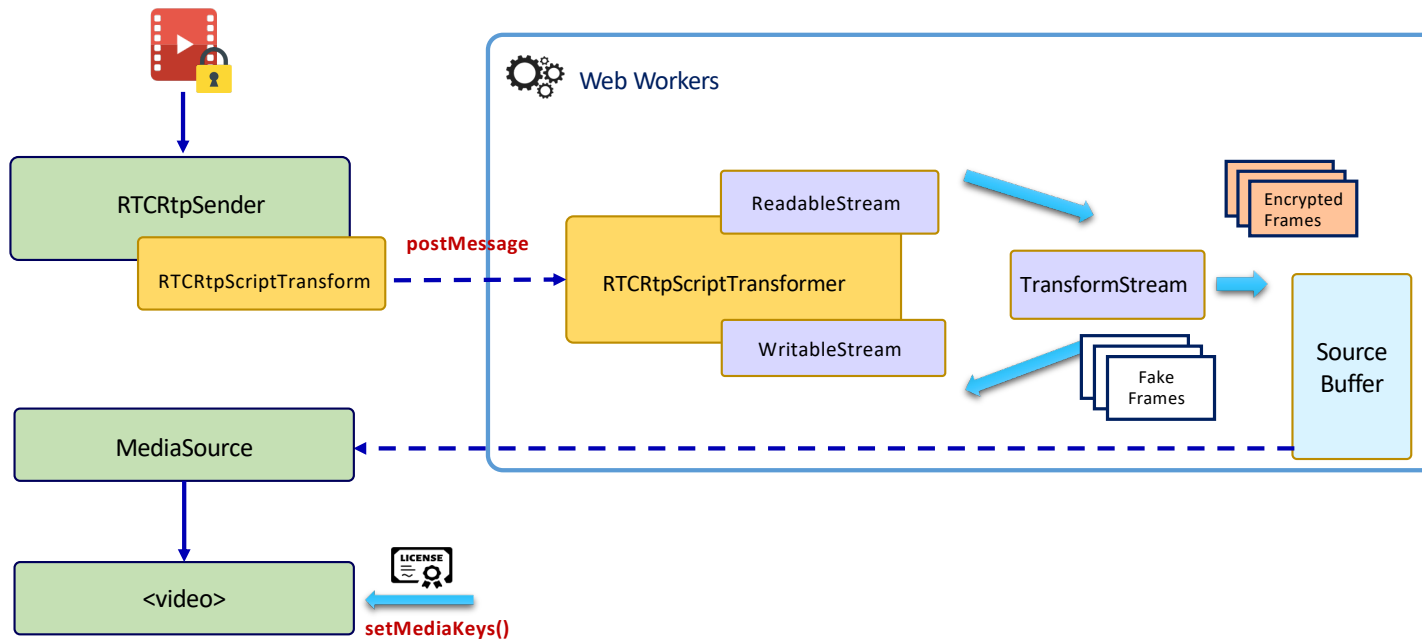
DRM requirements for WebRTC one-way media:

- Transmitting stored pre-encoded media as part of the WebRTC RTP session
- Decrypting media with CDM, and decoding pre-encoded media with MSE.



Case 2: Digital Rights Management

Current DRM solution for WebRTC



Case 3: H265 RTC Supporting

HEVC/H.265 decoding support on Web

| Chrome | Edge * | Safari | Firefox | Opera | IE | Chrome for Android | Safari on iOS * | Samsung Internet | Opera Mini * | Opera Mobile * |
|----------------------|---------------------|----------------------|----------------------|---------------------|-----------------|--------------------|-----------------|-------------------|--------------|-----------------|
| | | 3.1-10.1 | 2-119 | | | | | 4 | | |
| 4-106 | ¹ 12-18 | ³ 11-12.1 | ⁶ 120 | 10-93 | | | 3.2-10.3 | ² 5-20 | | |
| ⁵ 107-124 | ⁴ 79-123 | 13-17.4 | ⁷ 121-125 | ⁵ 94-108 | 6-10 | | 11-17.4 | 21-23 | | 12-12.1 |
| ⁵ 125 | ⁴ 124 | 17.5 | ⁷ 126 | ⁵ 109 | ¹ 11 | ⁵ 124 | 17.5 | 24 | all | ² 80 |
| ⁵ 126-128 | | 17.6-TP | ⁷ 127-129 | | | | 17.6 | | | |

- Supported only for devices with hardware support
- Reported to work in certain Android devices with hardware support
- Supported only on macOS High Sierra or later
- Supported for all devices on macOS (>= Big Sur 11.0) and Android (>= 5.0) if Edge >= 107, for devices with hardware support on Windows (>= Windows 10 1709) when HEVC video extensions from the Microsoft Store is installed
- Supported for all devices on macOS (>= Big Sur 11.0) and Android (>= 5.0),

- for devices with hardware support on Windows (>= Windows 8), and for devices with hardware support powered by VAAPI on Linux and ChromeOS
- Supported for devices with hardware support (the range is the same as Edge) on Windows in Nightly only. 10-bit or higher colors are not supported.
- Supported for devices with hardware support (the range is the same as Edge) on Windows only. Enabled by default in Nightly and can be enabled via the media.wmf.hevc.enabled pref in about:config. 10-bit or higher colors are not supported.

Case 3: H265 RTC Supporting

HEVC Encoding support on Web



Safari 13+ on Mac and IOS



Chrome for Windows / Mac M109+; Chrome for Android M117+;
Experimental feature with switch: `--enable-features=PlatformHEVCEncoderSupport`



Not supported

WebRTC HEVC Support



Experimental feature on **Safari 14+**
Not compatible with RFC 7798 Packetization



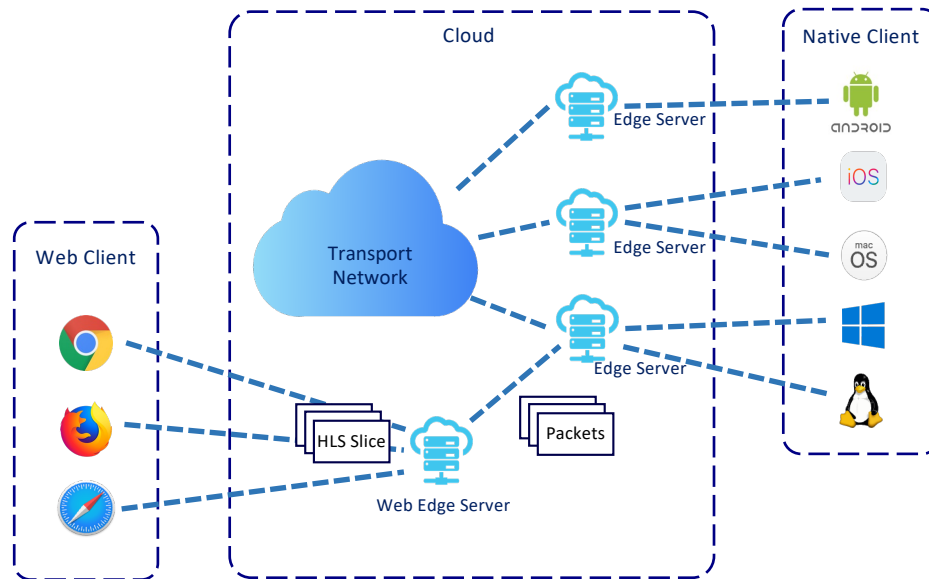
Chrome Canary 127
Experimental feature with switch: `--enable-features=PlatformHEVCEncoderSupport, WebRtcAllowH265Send, WebRtcAllowH265Receive --force-fieldtrials=WebRTC-Video-H26xPacketBuffer/Enabled`



Not supported

Case 3: H265 RTC Supporting

Solution 1: Remuxing RTP packets to HLS on Server



Pros

- The architecture is relatively simple
- The system has good compatibility

Cons

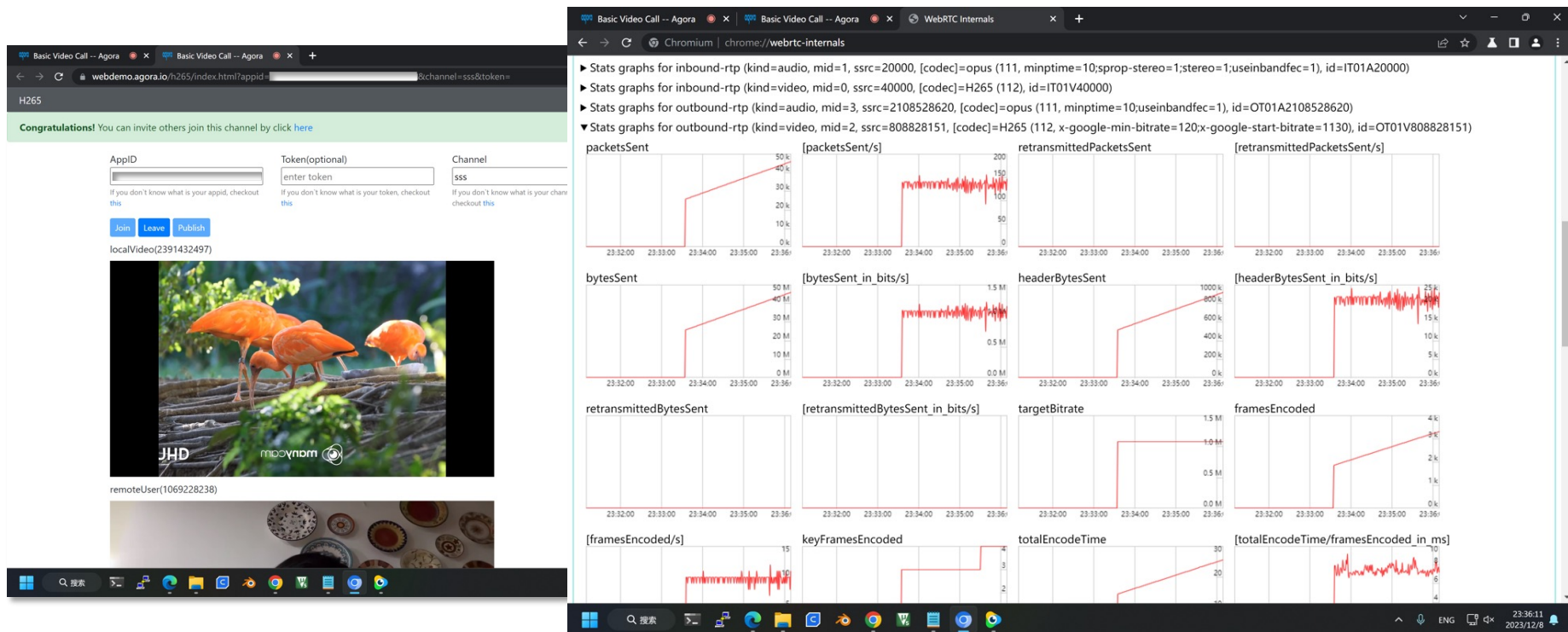
- Not a real-time system
- Poor resistance to weak network

Scenario

Good network quality, do not have strict real-time requirements.

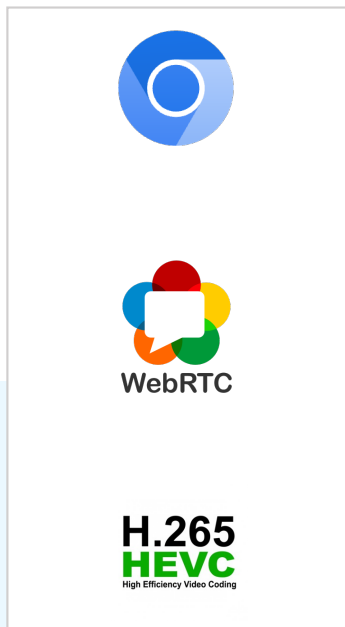
Case 3: H265 RTC Supporting

Solution 2: Customized Browser with WebRTC H265 Support



Case 3: H265 RTC Supporting

Solution 2: Customized Browser with WebRTC H265 Support



Pros

- Pure web technical stack for developer
- Compatible with existed WebRTC apps
- Forward compatibility with future official Chrome browser

Cons

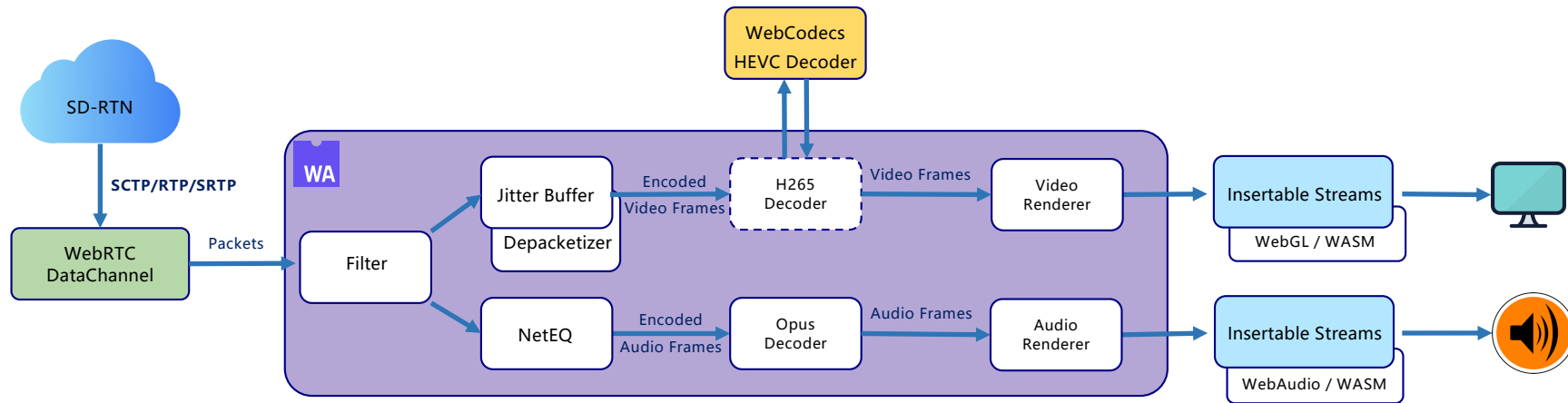
- Higher technical maintenance costs
- Extra efforts for client distribution

Scenario

Application environment under control in organization like company, bureau, etc...

Case 3: H265 RTC Supporting

Solution 3: Port RTC components to WebAssembly



Features

- Implement full downlink pipeline with WebAssembly. Including bandwidth estimation, jitter buffer, netEQ、video packetizer/depaketizer, media codecs, media renderer.
- Media is transmitted with tuned WebRTC DataChannel

Case 3: H265 RTC Supporting

Potential Future Solution: WebRTC-RtpTransport API

A new proposal being discussed in WebRTC WG

Problem & Motivation

WebRTC APIs is not sufficient, due to:

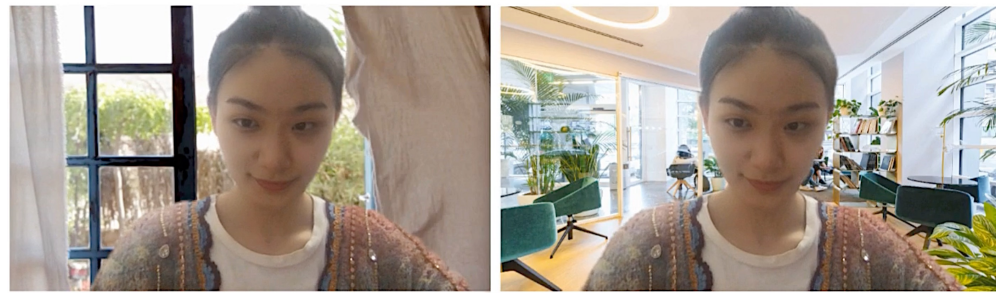
- Lack of support for custom metadata
 - Lack of codec support
 - Lack of custom rate control
- Inability to support custom RTCP messages

Goal

- Custom rate control (with built-in bandwidth estimate)
 - Custom bitrate allocation
- Custom metadata (header extensions)
 - Custom RTCP messages
 - Custom RTCP message timing
 - RTP forwarding
- Custom payloads (ML-based audio codecs)
 - Custom packetization
 - Custom FEC
 - Custom RTX
 - Custom Jitter Buffer
 - Custom bandwidth estimate

Case 4: Alpha Video Transmission

Scenario: Rendering characters onto virtual background



● Immersive online meeting



● Picture-in-picture presentation



● Virtual interaction

Case 4: Alpha Video Transmission

In WebRTC pipeline, alpha plane is ignored even if the encoder support alpha channel

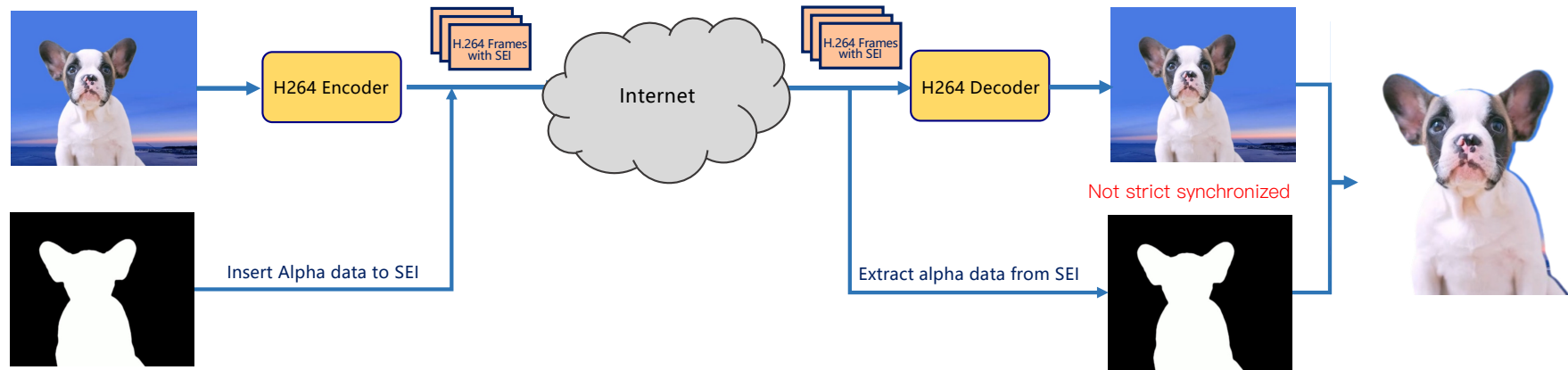
The screenshot displays a WebRTC application interface on the left and its source code on the right. The application shows a video call with a local video and a remote user. The source code is from Chromium, showing the libvp9_encoder.cc file. The code includes a switch statement for VideoFrameBuffer::Type, with cases for kI420 and kI420A. The kI420A case is highlighted, showing that the alpha plane is not processed. The code also shows the preparation of raw data from mapped buffers for VPX and NV12 formats.

```

2043 // Prepare `raw` from `mapped_buffer`.
2044 switch (mapped_buffer->type()) {
2045   case VideoFrameBuffer::Type:kI420:
2046     case VideoFrameBuffer::Type:kI420A: {
2047       MaybeRewrapRawWithFormat(VPX_IMG_FMT_I420);
2048       const I420BufferInterface* i420_buffer = mapped_buffer->GetI420();
2049       RTC_DCHECK(i420_buffer);
2050       raw->planes[VPX_PLANE_Y] = const_cast<uint8_t*>(i420_buffer->DataY());
2051       raw->planes[VPX_PLANE_U] = const_cast<uint8_t*>(i420_buffer->DataU());
2052       raw->planes[VPX_PLANE_V] = const_cast<uint8_t*>(i420_buffer->DataV());
2053       raw->stride[VPX_PLANE_Y] = i420_buffer->StrideY();
2054       raw->stride[VPX_PLANE_U] = i420_buffer->StrideU();
2055       raw->stride[VPX_PLANE_V] = i420_buffer->StrideV();
2056       break;
2057     }
2058   case VideoFrameBuffer::Type:knv12: {
2059     MaybeRewrapRawWithFormat(KNV12);
2060     const NV12BufferInterface* nv12_buffer = mapped_buffer->GetNV12();
2061     RTC_DCHECK(nv12_buffer);
2062     raw->planes[VPX_PLANE_Y] = const_cast<uint8_t*>(nv12_buffer->DataY());
2063     raw->planes[VPX_PLANE_U] = const_cast<uint8_t*>(nv12_buffer->DataUV());
2064     raw->planes[VPX_PLANE_V] = raw->planes[VPX_PLANE_U] + 1;
2065     raw->stride[VPX_PLANE_Y] = nv12_buffer->StrideY();
2066     raw->stride[VPX_PLANE_U] = nv12_buffer->StrideUV();
2067     raw->stride[VPX_PLANE_V] = nv12_buffer->StrideUV();
2068     break;
2069   }
2070   default:
2071     RTC_DCHECK_NOTREACHED();
2072 }
2073 return mapped_buffer;
2074 }
2075 }
2076 // namespace webrtc
2077 }
2078 #endif // RTC_ENABLE_VP9
  
```

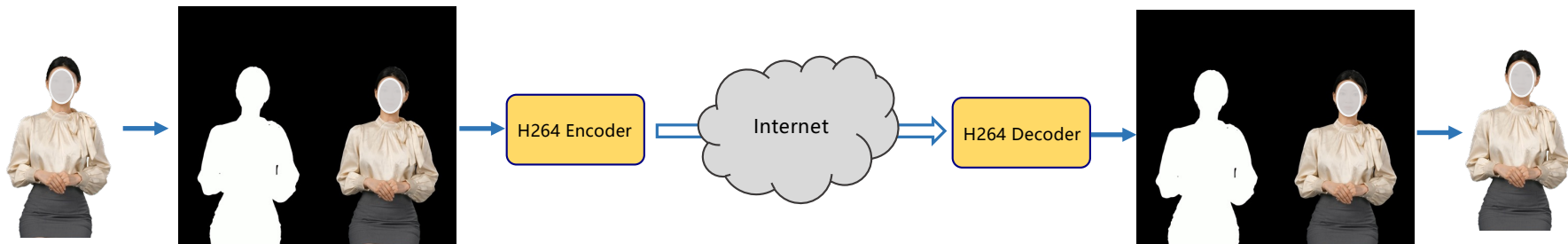

Case 4: Alpha Video Transmission

Solution 1: Send alpha data with H.264 SEI



Case 4: Alpha Video Transmission

Solution 1: Store alpha data to the expanded area



- Need extra metadata to record video type (normal or expanded)
- Cost about 20% extra bandwidth

END

Thanks