Cloud Streaming Services

SDK Guide

Product Documentation

Contents

SDK Guide

0. SDK Integration Guide

Last updated：2022-11-21 16:56:28

This document describes how to quickly integrate live push and stream pull solutions into your business program. The SDKs for different businesses are as listed below:

Feature Module		Mobile	Web	Documentation
Stream push		iOS / Android	Web	1. Stream Push
Playback		iOS / Android	-	2. Playback
Advanced features	AV1 video playback	AV1 Encoding		Overview
	High-quality upstreaming	TMIO SDK	-	
	LEB playback by player	Integrating libLebConnection SDK into a player	-	
	On-screen comments and chat	Integration guide	-	
	Beauty filters and special effects	iOS / Android	-	

2. Playback

Last updated：2022-11-21 16:56:28

Prerequisites

- You have activated the CSS service.
- Select Domain Management, click **Add Domain**, and add a playback domain name as instructed in Adding Your Own Domain.
- In the CSS console, generate a playback address in **CSS Toolkit** > **Address Generator** as instructed in Address Generator. Then, implement live playback in your own business based on your business scenarios as follows:

Integration into an application

Download and integrate the MLVB SDK as instructed in the iOS and Android integration guides.

Note：
Enable stream pull and playback. Below is the configuration for iOS and Android:

- iOS

```
V2TXLivePlayer *_txLivePlayer = [[V2TXLivePlayer alloc] init];
>
```

- Android

```
V2TXLivePlayer mLivePlayer = new V2TXLivePlayerImpl(mContext);
```

More

- Using the MLVB SDK will incur fees. For billing details, see Billing Overview.

3. Advanced Features
Overview

Last updated：2022-11-21 16:56:28

This document describes how to import advanced live streaming features into your program.

AV1 Video Playback

AV1 is an open-source and royalty-free video compression format. Compared with its predecessor H.265 (HEVC) encoding, it can further reduce the bitrate by over 30% while maintaining the same level of image quality. This means that it can deliver a higher image quality at the same bandwidth. Currently, CSS supports AV1 encoding. However, to play back AV1 videos, your player must support decoding the AV1 format.
You can implement AV1 decoding in your own player as follows:

Container format and distribution protocol

Tencent Cloud has implemented a proprietary extension to AV1 in FLV in T-FFmpeg. To modify your player, you can extend the code based on the patch files of T-FFmpeg. For more information, see tencentyun/AV1.

Decoding

- **Hardware decoding**
  On PC, almost all new models of AMD, Intel, and Nvidia GPUs support AV1 hardware decoding.
  Device models supporting AV1 hardware decoding are as listed below:

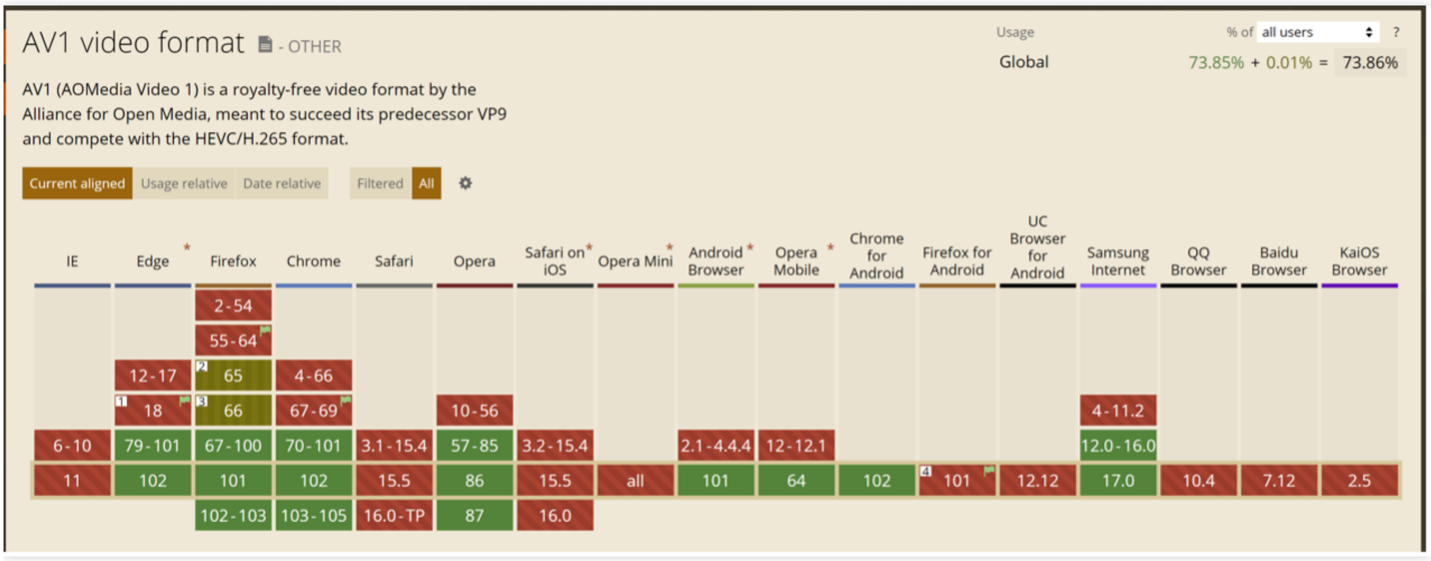
Type	Brand	Processor
Mobile phone	Realme X7 Pro	Dimensity 1000+
	OPPO Reno5 Pro	Dimensity 1000+
	HONOR V40	Dimensity 1000+
	Redmi K30 Ultra	Dimensity 1000+
	Vivo iQOO Z1	Dimensity 1000+
	Redmi Note 10 Pro	Dimensity 1000+
	Vivo S9	Dimensity 1100

Type	Brand	Processor
	Realme Q3 Pro	Dimensity 1100
	Vivo S10	Dimensity 1100
	Vivo S12	Dimensity 1100
	Vivo S12 Pro	Dimensity 1200
	OPPO Reno6 Pro	Dimensity 1200
	OPPO Reno7 Pro	Dimensity 1200
	Redmi K40 Pro	Dimensity 1200
	Realme GT NEO	Dimensity 1200
	HONOR X20	Dimensity 1200
	OnePlus Nord 2	Dimensity 1200
	Realme GT NEO 2	Dimensity 1200
	OPPO K9 Pro	Dimensity 1200
	OPPO Find X5 Pro Dimensity Edition	Dimensity 9000
	Redmi K50	Dimensity 9000
	Galaxy S21 (Exynos Edition)	Exynos 2100
	Galaxy S22 (Exynos Edition)	Exynos 2200
TV	Samsung Q950TS QLED 8K TV	-

- **Software decoding**
  - av1d (Tencent Cloud's optimized AV1 decoder, which outperforms dav1d and can provide closed-source libraries. You can integrate it as instructed in av1d Integration Guide. T-FFmpeg provides FFmpeg patches to be integrated and av1d libraries.)
  - dav1d
  - libgav1
  - Android 10.0 integrates an AV1 decoder.
  - The Chrome family supports AV1 decoding.

Support by browsers

The Chrome family supports AV1 decoding while browsers on iOS don't.



AV1 video format - OTHER

Usage: Global — % of all users: 73.85% + 0.01% = 73.86%

AV1 (AOMedia Video 1) is a royalty-free video format by the Alliance for Open Media, meant to succeed its predecessor VP9 and compete with the HEVC/H.265 format.

Current aligned | Usage relative | Date relative | Filtered | All

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2-54														
		55-64														
	12-17	2 65	4-66													
	1 18	3 66	67-69		10-56								4-11.2			
6-10	79-101	67-100	70-101	3.1-15.4	57-85	3.2-15.4		2.1-4.4.4	12-12.1				12.0-16.0			
11	102	101	102	15.5	86	15.5	all	101	64	102	4 101	12.12	17.0	10.4	7.12	2.5
		102-103	103-105	16.0-TP	87	16.0										

Note：
The above information was collected from this website in July 2022. To view the latest information, please visit the website.

TMIO SDK

The TMIO SDK customizes, encapsulates, and optimizes streaming media protocols such as SRT and QUIC. It safeguards upstreaming and implements low-latency transfer with a high packet loss prevention rate, multi-linkage transfer optimization, and the retransmission timeout (RTO) mechanism. It promises wide application in scenarios that require a high data source stability and long-range transfer.

Feature description

- The TMIO SDK is suitable for long-range transfer and the radio and TV industry.
- The TMIO SDK supports mainstream platforms, including Android, iOS, Windows, macOS, and Linux.

Integration method

You can integrate the SDK as instructed in Integration Steps.

libLebConnection SDK

The libLebConnection SDK provides upgraded transfer capabilities based on the native WebRTC. It allows you to connect your existing player to LEB with just a few simple modifications. Based on LVB-compatible stream push and cloud-based media processing capabilities, it can implement live streaming at a low latency even under high concurrency, so as to help you smoothly migrate from standard LVB streaming to low-latency LEB streaming. For large rooms in modern real-time communication (RTC) scenarios, it can also help you quickly implement relayed live streaming at low costs and low latency.

Feature description

- The libLebConnection SDK can pull audio/video streams at a low latency even under poor network conditions.
- It can play back H.264. H.265, and AV1 videos with B frames and output them as raw video frame data such as Annex B and OBU files for H.264/H.265 and AV1 input videos respectively.
- It can play back AAC and Opus audios and output them as raw audio frame data.
- It supports Android, iOS, Windows, Linux, and macOS.

Integration method

You can integrate the libLebConnection SDK as instructed in Integrating libLebConnection SDK into a Player.

Beauty filters and special effects

To integrate beauty filters and special effects and import beauty filters, image filters, and stickers during live streaming, you can integrate the Tencent Effect SDK.

Integration into an application

Download the Tencent Effect SDK here and integrate it as instructed in the iOS and Android integration guides.

More

Using the Tencent Effect SDK will incur fees. For billing details, see Pricing Overview.

TMIO SDK

Last updated：2023-11-24 15:01:14

Overview

As more and more streaming protocols emerge, it has become a challenge to build an application that supports all protocols. For this, we have developed the Tencent Media IO (TMIO) SDK, which is optimized for mainstream protocols on the market, to help you build stable and highly available media applications with ease.
The TMIO SDK supports mainstream protocols including SRT and QUIC, as well as Tencent's proprietary protocol, Elastic Transmission Control (ETC). More will be added in the future.

Strengths

Multi-platform support: The SDK comes in editions for Android, iOS, Linux, macOS, and Windows.
Different integration methods:
You can use the non-intrusive proxy mode to use the SDK, with no code changes required.
The simple API design also allows you to quickly integrate the SDK into your application, replacing your existing protocols.
Simple API design with high compatibility and flexibility:
Easy-to-use APIs are offered.
You can choose different modes and policies based on your business scenario.
The SDK can be extended to support other protocols.
Multi-protocol support and optimization:
The SDK supports emerging streaming protocols including SRT and QUIC, as well as Tencent's proprietary ETC protocol.
It can meet many requirements in different business scenarios.
It delivers low-latency, secure, and reliable transfer based on UDP.
It supports multi-connection acceleration, which guarantees transfer stability and smoothness.

Test (TMIO SRT)

The TMIO SDK supports the SRT protocol, which can improve upstream stability and downstream smoothness under poor network conditions or in long-distance transfer scenarios.
In the test below, with RTMP streaming, playback begins to stutter when packet loss reaches 10%, but for SRT streaming, stability and low latency are guaranteed even with a packet loss of 30%.

Features

SRT-based streaming media transfer

High tolerance for random packet loss

A retransmission mechanism based on ARQ and a timeout policy

Low-latency, secure, and reliable transfer based on UDT

Multi-connection transfer and the aggregation transfer mode:

You can configure multiple connections to transfer data. For example, mobile devices can transfer data over both Wi-Fi and the 4G/5G data network, so even if one of the connections is disconnected, data transfer will not be affected. This improves transfer reliability.

Transfer Mode	Description
Broadcasting mode	You can send redundant data over multiple internet connections to ensure data integrity and transfer reliability.
Primary/Backup mode	In this mode, only one connection is active at a time. The connection is selected in real time based on stability and reliability. This also reduces bandwidth usage because no redundant data is sent.
Aggregation mode	In scenarios requiring a high bitrate and bandwidth, the bandwidth of a single internet connection may be unable to meet the requirements. This mode can split data, send them over multiple connections, and then combine them at the receiver end.

QUIC-based media transfer

Adaptive congestion control algorithm

Smooth network switch

Support for the next-generation HTTP/3

Less redundant data sent when bandwidth is limited or network is unstable

Proprietary streaming protocol ETC

Innovative, lightweight, and cross-platform

Support for IoT devices (peer-to-peer communication)

Quick startup, low latency, and efficient bandwidth utilization

Quick and accurate detection of internet connection changes to ensure that the optimal transfer policy is always used

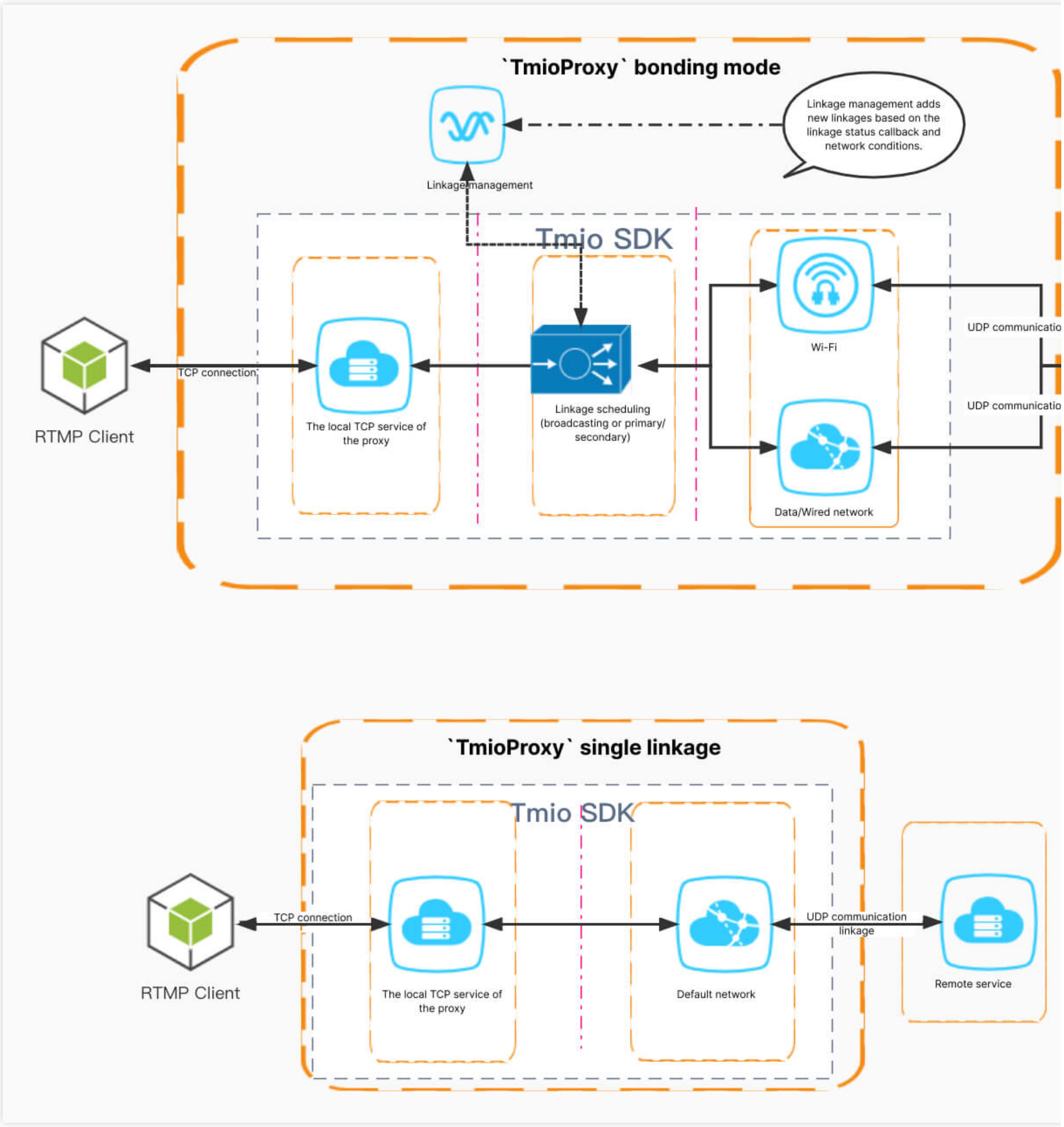
Fair and stable usage of bandwidth when used together with mainstream streaming protocols

Integration Directions

The directions below use the RTMP over SRT protocol as an example.

Using the proxy mode

Workflow


`TmioProxy` bonding mode
Linkage management adds new linkages based on the linkage status callback and network conditions.
Linkage management
Tmio SDK
RTMP Client
TCP connection
The local TCP service of the proxy
Linkage scheduling (broadcasting or primary/ secondary)
Wi-Fi
Data/Wired network
UDP communicatio
UDP communicatio
`TmioProxy` single linkage
Tmio SDK
RTMP Client
TCP connection
The local TCP service of the proxy
Default network
UDP communication linkage
Remote service

Directions

1. **Create a `TmioProxy` instance**:

```
std::unique_ptr<tmio::TmioProxy> proxy_ = tmio::TmioProxy::createUnique();
```

2. **Set the listener**:

```
void setListener(TmioProxyListener *listener);
```

Below are the callbacks of `TmioProxyListener` :

TMIO configuration callback

TMIO proxy startup callback

Error callback

You can configure `Tmio` parameters in this callback. **For simple configuration, you can use the auxiliary methods provided in `tmio-preset.h` .**

```
/*
void onTmioConfig(Tmio *tmio);
*/
void onTmioConfig(tmio::Tmio *tmio) override {
        auto protocol = tmio->getProtocol();
        if (protocol == tmio::Protocol::SRT) {
                tmio::SrtPreset::rtmp(tmio);
        } else if (protocol == tmio::Protocol::RIST) {
                tmio->setIntOption(tmio::base_options::RECV_SEND_FLAGS,
                                                     tmio::base_options::FLAG_SEND)
        }
}
```

```
/*
void onStart(const char *local_addr, uint16_t local_port);
*/
void onStart(const char *addr, uint16_t port) override {
        LOGFI("ip %s, port %" PRIu16, addr, port);
}
```

This callback indicates that the remote server is connected successfully, and the local TCP port is bound successfully. You can start pushing streams.

```
/*
void onError(ErrorType type, const std::error_code &err);
*/
void onError(tmio::TmioProxyListener::ErrorType type,
                        const std::error_code &err) override {
        LOGFE("error type %s, %s, %d", tmio::TmioProxyListener::errorType(type),
                    err.message().c_str(), err.value());
}
```

You can use `ErrorType` to determine whether an error is a local or remote I/O error. A local I/O error is usually because RTMP streaming is stopped by the streamer. Therefore, if streaming has ended, you can ignore such errors. However, a remote I/O error usually needs to be handled.

3. **Start the proxy**:

```
std::error_code start(const std::string &local_url, const std::string &remote_url,
```

API parameters

Parameter	Description

local_url	It supports only the TCP scheme in the format of tcp://${ip}:${port}. `port` can be `0`, which indicates to bind a random port. After the binding succeeds, the bound port will be returned to the application through the `onStart()` callback. Using port `0` can avoid binding failures due to issues such as port occupancy and no permissions.
remote_url	The remote server URL.
config	The configuration parameter. This parameter is valid if SRT connection bonding or QUIC H3 is enabled. For details, see the definition of the `SrtFeatureConfig` structure in tmio.h.

The sample code for a single connection:

```
proxy_->start(local_url, remote_url, NULL);
```

The sample code for multi-connection bonding:

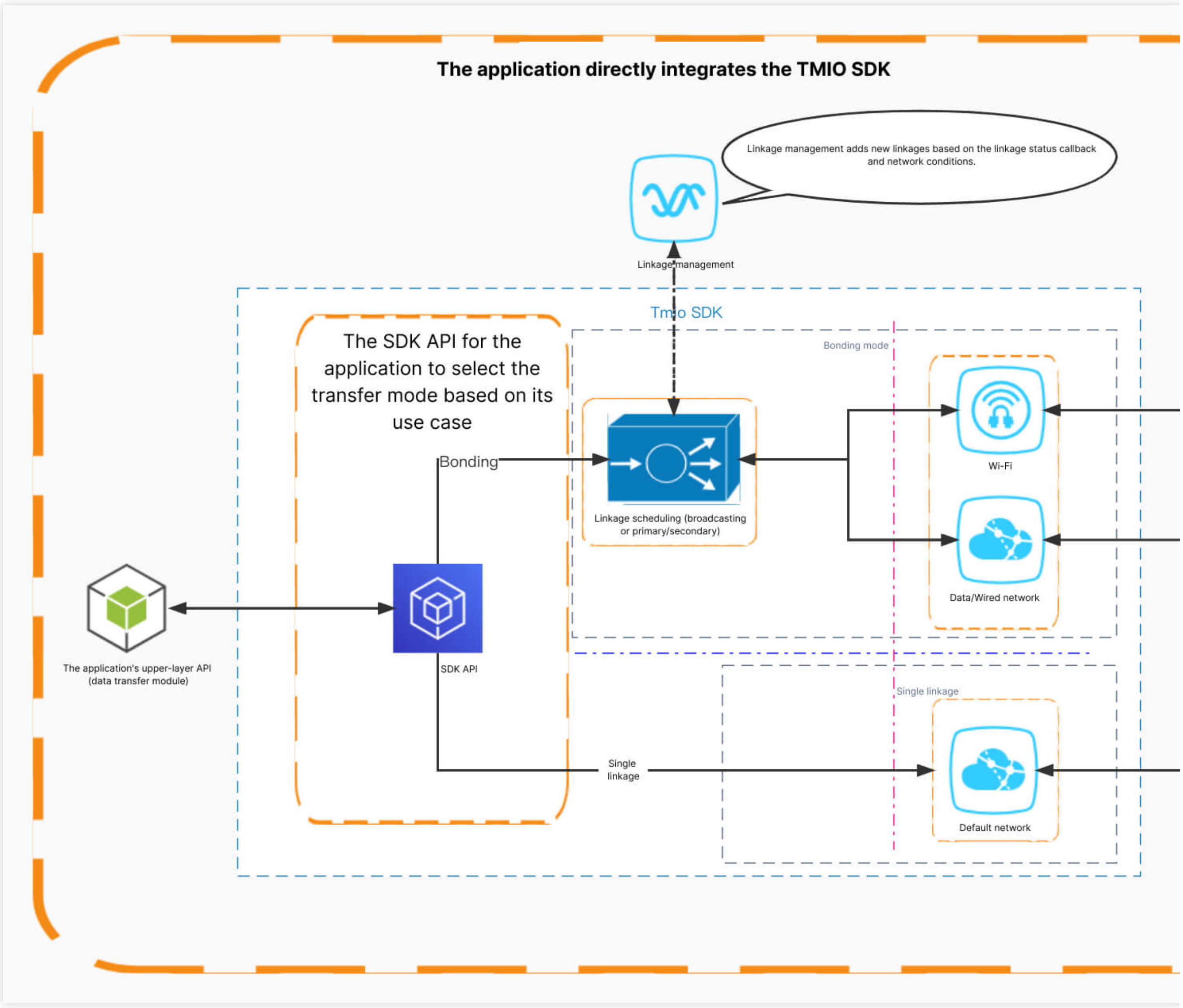
```
tmio::TmioFeatureConfig option;
option_.protocol = tmio::Protocol::SRT;
option_.trans_mode = static_cast<int>(tmio::SrtTransMode::SRT_TRANS_BACKUP);
/*---------------------------------------------------------------*/
{
// You can add multiple connections as needed.
option_.addAvailableNet(net_name, local_addr, remote_url, 0, weight, -1);
}
/*-----------------------------------------------------------*/

 proxy_->start(local_url, remote_url, &option_);
```

4. **Stop the proxy**:

```
/*
void stop();
*/
proxy_.stop();
```

Integrating the TMIO SDK into your application

Workflow


The application directly integrates the TMIO SDK
Linkage management adds new linkages based on the linkage status callback and network conditions.
Linkage management
Tmio SDK
The SDK API for the application to select the transfer mode based on its use case
Bonding mode
Bonding
Linkage scheduling (broadcasting or primary/secondary)
Wi-Fi
Data/Wired network
The application's upper-layer API (data transfer module)
SDK API
Single linkage
Single linkage
Default network

Directions

1. **Create a `Tmio` instance and configure the parameters**:

```
tmio_ = tmio::TmioFactory::createUnique(tmio::Protocol::SRT);
tmio::SrtPreset::mpegTsLossless(tmio_.get());
tmio_->setIntOption(tmio::srt_options::CONNECT_TIMEOUT, 4000);
tmio_->setBoolOption(tmio::base_options::THREAD_SAFE_CHECK, true);
```

Create a `Tmio` instance: You can use `TmioFactory` to create it.

Configure parameters: Select different APIs to configure the parameters:

Parameters: See tmio-option.h.

Simple configuration: See tmio-preset.h.

```
// Select an appropriate configuration based on different parameter attributes
bool setBoolOption(const std::string &optname, bool value);
bool setIntOption(const std::string &optname, int64_t value);
bool setDoubleOption(const std::string &optname, double value);
bool setStrOption(const std::string &optname, const std::string &value);
...
```

2. **Start connection**:

```
/**
 * open the stream specified by url
 *
 * @param config protocol dependent
 */
virtual std::error_code open(const std::string &url,
                             void *config = nullptr) = 0;
```

A single connection (`config` can be `NULL`)

```
// A single connection by default
auto err = tmio->open(TMIO_SRT_URL);
if (err) {
LOGE("open failed, %d, %s", err.value(), err.message().c_str());
}
```

Multi-connection bonding (currently, only the SRT protocol is supported)

For how to set `config` for the SRT bonding feature, see the definition of the `TmioFeatureConfig` structure in the `tmio.h` file.

```
tmio::TmioFeatureConfig option_;
option_.protocol = tmio::Protocol::SRT;
option_.trans_mode = static_cast<int>(tmio::SrtTransMode::SRT_TRANS_BACKUP);
option_.addAvailableNet(net_name, local_addr, remote_url, 0, weight, -1);
```

```
// Multi-connection bonding
auto err = tmio_->open(TMIO_SRT_URL, &option_);
if (err) {
 LOGE("open failed, %d, %s", err.value(), err.message().c_str());
}
```

With multi-connection bonding, you can use the `open` API to add new transfer connections to the group.

3. **Send data**:

```
int ret = tmio_->send(buf.data(), datalen, err);
if (ret < 0) {
        LOGE("send failed, %d, %s", err.value(), err.message().c_str());
        break;
}
```

4. **Receive data**:

For protocols that involve interactions, such as RTMP, you need to call an API to read the data. We offer two APIs for this:

```
/**
* receive data
*
* @param err return error details
* @return number of bytes which were received, or < 0 to indicate error
*/
virtual int recv(uint8_t *buf, int len, std::error_code &err) = 0;

using RecvCallback = std::function<bool(const uint8_t *buf, int len, const std::err
/**
* receive data in event loop
```

```
 *
 * recvLoop() block current thread, receive data in a loop and pass the data to recv
 * @param recvCallback return true to continue the receive loop, false for break
 */
virtual void recvLoop(const RecvCallback &recvCallback) = 0;
```

Loop read of the upper-layer application:

```
    while (true) {
  ret = tmio_->recv(buf.data(), buf.size(), err);
  if (ret < 0) {
    LOGE("recv error: %d, %s", err.value(), err.message().c_str());
```

```
        break;
    }
    ...
}
```

Read through callback:

```
FILE *file = fopen(output_path, "w");
tmio_->recvLoop([file](const uint8_t *buf, int len,
                                                const std::error_code &err) {
```

```
    if (len < 0) {
            fwrite(buf, 1, len, file);
    } else if (len < 0) {
            LOGE("recv error: %d, %s", err.value(), err.message().c_str());
    }
    return true;
});
```

5. **Terminate the `Tmio` instance**:

```
tmio_->close();
```

6. **More**:

Get the current connection status (your application can adjust its stream push policy based on the status):

```
tmio::PerfStats stats_;
tmio_->control(tmio::ControlCmd::GET_STATS, &stats_);
```

API and Demo Updates

To get the latest updates on the APIs and demos of the SDK, visit this GitHub page.

FAQs

Is SRT multi-connection bonding supported for all devices?

Only devices with multiple available network interfaces can use multi-connection bonding, and Android devices can use this feature only if the Android version is 6.0 or later and the API level is 23 or above.

How do I enable the 4G/5G data network on an Android phone connected to Wi-Fi?

An Android phone connected to Wi-Fi cannot transfer data directly over the 4G/5G network. To enable the data network, apply for the data network permission as follows:

```
ConnectivityManager connectivityManager = (ConnectivityManager) context.getSystemSe
NetworkRequest request = new NetworkRequest.Builder().addTransportType(NetworkCapab
                                           .addCapability(NetworkCapabilities.NET_C
                                           .build();

ConnectivityManager.NetworkCallback networkCallback = new ConnectivityManager.Netwo
    @Override
    public void onAvailable(@NonNull Network network) {
                Log.d(TAG, "The mobile data network has been enabled");
        super.onAvailable(network);
    }
}
```

Integrating libLebConnection SDK into a Player

Last updated：2022-12-23 10:56:16

Overview

The libLebConnection SDK provides upgraded transfer capabilities based on the native WebRTC. It allows you to connect your existing player to LEB with just a few simple modifications. Based on LVB-compatible stream push and cloud-based media processing capabilities, it can implement live streaming at a low latency even under high concurrency, so as to help you smoothly migrate from standard LVB streaming to low-latency LEB streaming. For large rooms in modern real-time communication (RTC) scenarios, it can also help you quickly implement relayed live streaming at low costs and low latency.

Feature Description

- The libLebConnection SDK can pull audio/video streams at a low latency even under poor network conditions.
- It can play back H.264. H.265, and AV1 videos with B frames and output them as raw video frame data such as Annex B and OBU files for H.264/H.265 and AV1 input videos respectively.
- It can play back AAC and Opus audios and output them as raw audio frame data.
- It supports Android, iOS, Windows, Linux, and macOS.

Integration Method

Basic API description

- Create a LEB connection.

```
LEB_EXPORT_API LebConnectionHandle* OpenLebConnection(void* context, LebLogLeve
l loglevel);
```

- Register a callback function.

```
LEB_EXPORT_API void RegisterLebCallback(LebConnectionHandle* handle, const LebC
allback* callback);
```

- Start the connection to pull the stream.

```
LEB_EXPORT_API void StartLebConnection(LebConnectionHandle* handle, LebConfig c
onfig);
```

- Stop the connection.

```
LEB_EXPORT_API void StopLebConnection(LebConnectionHandle* handle);
```

- Close the connection.

```
LEB_EXPORT_API void CloseLebConnection(LebConnectionHandle* handle);
```

Callback API description

```
typedef struct LebCallback {
// The log callback
OnLogInfo onLogInfo;
// The video information callback
OnVideoInfo onVideoInfo;
// The audio information callback
OnAudioInfo onAudioInfo;
// The video data callback
OnEncodedVideo onEncodedVideo;
// The audio data callback
OnEncodedAudio onEncodedAudio;
// The `MetaData` callback
OnMetaData onMetaData;
// The statistics callback
OnStatsInfo onStatsInfo;
// The error callback
OnError onError;
} LebCallback;
```

Note：
For the detailed definitions of data structures, see `leb_connection_api.h` .

API call process

1. **Create an LEB connection**: `OpenLebConnection()`
2. **Register various callback functions**: `RegisterXXXXCallback()`
3. **Start the connection to pull the stream**: `StartLebConnection()`
4. **Call back and output the raw audio/video data**:

- OnEncodedVideo()
- OnEncodedAudio()

5. **Stop the connection**: `StopLebConnection()`
6. **Close the connection**: `CloseLebConnection`

Integration sample

This sample describes how to integrate libLebConnection into the typical open-source player ijkplayer widely used on Android devices. For how to integrate it on other platforms, you can also refer to the sample code.

Latest SDK version

You can download the libLebConnection SDK here.

FAQs on Integration

How do I develop the lag statistics collection feature?

As buffering is disabled, you can now collect lag statistics based on the video rendering interval. If the video rendering interval exceeds a certain threshold, it will be counted as a lag, and the lag duration will be added to the total lag duration.

Taking ijkplayer as an example, you can develop the lag statistics feature as follows:

1. **Modify the code**
   i. Add the variables required by lag statistics collection to the `VideoState` and `FFPlayer` structures.

```
diff --git a/ijkmedia/ijkplayer/ff_ffplay_def.h b/ijkmedia/ijkplayer/ff_ffplay_de
f.h
index 00f19f3c..f38a790c 100755
--- a/ijkmedia/ijkplayer/ff_ffplay_def.h
+++ b/ijkmedia/ijkplayer/ff_ffplay_def.h
@@ -418,6 +418,14 @@ typedef struct VideoState {
SDL_cond *audio_accurate_seek_cond;
volatile int initialized_decoder;
int seek_buffering;
+
+ int64_t stream_open_time;
+ int64_t first_frame_display_time;
+ int64_t last_display_time;
+ int64_t current_display_time;
+ int64_t frozen_time;
+ int frozen_count;
+ float frozen_rate;
} VideoState;
/* options specified by the user */
@@ -720,6 +728,14 @@ typedef struct FFPlayer {
char *mediacodec_default_name;
int ijkmeta_delay_init;
int render_wait_start;
int low_delay_playback;
+ int frozen_interval;
int high_level_ms;
int low_level_ms;
int64_t update_plabyback_rate_time;
int64_t update_plabyback_rate_time_prev;
} FFPlayer;
#define fftime_to_milliseconds(ts) (av_rescale(ts, 1000, AV_TIME_BASE))
@@ -844,6 +860,15 @@ inline static void ffp_reset_internal(FFPlayer *ffp)
ffp->pf_playback_volume = 1.0f;
ffp->pf_playback_volume_changed = 0;
ffp->low_delay_playback = 0;
ffp->high_level_ms = 500;
ffp->low_level_ms = 200;
+ ffp->frozen_interval = 200;
ffp->update_plabyback_rate_time = 0;
ffp->update_plabyback_rate_time_prev = 0;
av_application_closep(&ffp->app_ctx);
ijkio_manager_destroyp(&ffp->ijkio_manager_ctx);
```

ii. **Add the logic for lag statistics collection**

```
diff --git a/ijkmedia/ijkplayer/ff_ffplay.c b/ijkmedia/ijkplayer/ff_ffplay.c
index 714a8c9d..c7368ff5 100755
--- a/ijkmedia/ijkplayer/ff_ffplay.c
+++ b/ijkmedia/ijkplayer/ff_ffplay.c
@@ -874,6 +874,25 @@ static void video_image_display2(FFPlayer *ffp)
VideoState *is = ffp->is;
Frame *vp;
Frame *sp = NULL;
+ int64_t display_interval = 0;
+
+ if (!is->first_frame_display_time){
+ is->first_frame_display_time = SDL_GetTickHR() - is->stream_open_time;
+ }
+
+ is->last_display_time = is->current_display_time;
+ is->current_display_time = SDL_GetTickHR() - is->stream_open_time;
+ display_interval = is->current_display_time - is->last_display_time;
+ av_log(NULL, AV_LOG_DEBUG, "last_display_time:%"PRId64" current_display_tim
e:%"PRId64" display_interval:%"PRId64"\n", is->last_display_time, is->current_dis
play_time, display_interval);
+
+ if (is->last_display_time > 0) {
+ if (display_interval > ffp->frozen_interval) {
+ is->frozen_count += 1;
+ is->frozen_time += display_interval;
+ }
+ }
+ is->frozen_rate = (float) is->frozen_time / is->current_display_time;
+ av_log(NULL, AV_LOG_DEBUG, "frozen_interval:%d frozen_count:%d frozen_time:%"PR
Id64" is->current_display_time:%"PRId64" frozen_rate: %f ", ffp->frozen_interval,
is->frozen_count, is->frozen_time, is->current_display_time, is->frozen_rate);
vp = frame_queue_peek_last(&is->pictq);
```

Note：

In this sample, the initial value of the lag threshold (`frozen_interval`) is `200(ms)` , which can be adjusted as needed.

2. **Test lag statistics collection**

   Use QNET to simulate poor network conditions for the test as follows:

   i. Download QNET here.

   ii. Open QNET, click **Add** > **Template Type** > **Custom Template** and configure a template and parameters for poor network conditions as needed (the following is 30% random network packet loss during downstreaming).

   iii. Select a program from the program list.

   iv. Enable the configuration of poor network conditions for testing.

Note：

To facilitate testing, you can modify the above lag parameters and deliver the data to the Java layer through JNI to display it.

How do I eliminate the noise during playback? (optimization of SoundTouch for Android)

To adjust the playback rate based on the buffer watermarks, you need to use SoundTouch to adjust the audio speed. However when many network fluctuations occur, multiple speed adjustments are required as the buffer watermarks are frequently adjusted, which may cause noise when the native ijkplayer calls SoundTouch. In this case, you can refer to the following code for optimization:

When SoundTouch is called for speed adjustment in low-latency playback mode, all the buffer is translated by SoundTouch.

```
diff --git a/ijkmedia/ijkplayer/ff_ffplay.c b/ijkmedia/ijkplayer/ff_ffplay.c
index 714a8c9d..c7368ff5 100755
--- a/ijkmedia/ijkplayer/ff_ffplay.c
+++ b/ijkmedia/ijkplayer/ff_ffplay.c
@@ -2579,7 +2652,7 @@ reload:
int bytes_per_sample = av_get_bytes_per_sample(is->audio_tgt.fmt);
resampled_data_size = len2 * is->audio_tgt.channels * bytes_per_sample;
#if defined(__ANDROID__)
- if (ffp->soundtouch_enable && ffp->pf_playback_rate != 1.0f && !is->abort_reque
st) {
+ if (ffp->soundtouch_enable && (ffp->pf_playback_rate != 1.0f || ffp->low_delay_
playback) && !is->abort_request) {
av_fast_malloc(&is->audio_new_buf, &is->audio_new_buf_size, out_size * translate_
time);
for (int i = 0; i < (resampled_data_size / 2); i++)
{
```

Why can't MediaCodec decode H.265 videos after it is enabled?

The libLebConnection SDK supports H.265 video streams, but if you enable `Using MediaCodec` in **Settings** in the native ijkplayer, H.265 video streams won't be decoded by MediaCodec. In this case, you can refer to the following code for optimization:

```
diff --git a/ijkmedia/ijkplayer/ff_ffplay_options.h b/ijkmedia/ijkplayer/ff_ffpla
y_options.h
index b021c26e..958b3bae 100644
--- a/ijkmedia/ijkplayer/ff_ffplay_options.h
+++ b/ijkmedia/ijkplayer/ff_ffplay_options.h
@@ -178,8 +178,8 @@ static const AVOption ffp_context_options[] = {
OPTION_OFFSET(vtb_handle_resolution_change), OPTION_INT(0, 0, 1) },

// Android only options
- { "mediacodec", "MediaCodec: enable H.264 (deprecated by 'mediacodec-avc')",
- OPTION_OFFSET(mediacodec_avc), OPTION_INT(0, 0, 1) },
+ { "mediacodec", "MediaCodec: enable all_videos (deprecated by 'mediacodec_all_v
ideos')",
+ OPTION_OFFSET(mediacodec_all_videos), OPTION_INT(0, 0, 1) },
{ "mediacodec-auto-rotate", "MediaCodec: auto rotate frame depending on meta",
OPTION_OFFSET(mediacodec_auto_rotate), OPTION_INT(0, 0, 1) },
{ "mediacodec-all-videos", "MediaCodec: enable all videos",
```

WebRTC-Based Local Stream Mix

Last updated：2023-11-29 16:55:23

The SDK provides various video stream image processing features, including mixing multiple video streams (such as PiP), processing image effects (such as mirroring and filters), and adding other elements (such as watermarks and text). The basic process is as follows: The SDK first collects multiple streams and mixes them locally to combine the images and mix the audios. Then, it processes other effects. As all the features rely on the browser's support, the SDK has certain requirements for the browser performance. For the specific API protocols, see TXVideoEffectManager. This document describes the basic usage of local stream mix.

Basic Usage

To use the local stream mix feature, you should initialize the SDK and get the SDK instance `livePusher` . For the initialization code, see WebRTC Push.

Step 1. Get the video effect management instance

```
var videoEffectManager = livePusher.getVideoEffectManager();
```

Step 2. Enable local stream mix

First, you need to enable the local stream mix feature. By default, the SDK collects only one video stream and audio stream. After this feature is enabled, the SDK can collect multiple streams, which will be mixed locally by the browser.

```
videoEffectManager.enableMixing(true);
```

Step 3. Set the stream mix parameters

Set the stream mix parameters, especially the resolution and frame rate of the video output after stream mix.

```
videoEffectManager.setMixingConfig({
    videoWidth: 1280,
    videoHeight: 720,
    videoFramerate: 15
});
```

Step 4. Collect multiple streams

After local stream mix is enabled, the SDK starts collecting multiple streams such as the camera and shared screen. Be sure to keep the stream IDs, as they are required in subsequent operations.

```
var cameraStreamId = null;
var screenStreamId = null;

livePusher.startCamera().then((streamId) => {
    cameraStreamId = streamId;
}).catch((error) => {
    console.log('Failed to turn on the camera:'+ error.toString());
});

livePusher.startScreenCapture().then((streamId) => {
    screenStreamId = streamId;
}).catch((error) => {
    console.log('Failed to share the screen:'+ error.toString());
});
```

Step 5. Set the image layout

Set the layout of the images of the collected two streams. Here, the shared screen is displayed as the main image, with the camera image in the top-left corner. For the specific parameter configuration, see TXLayoutConfig.

```
videoEffectManager.setLayout([{
    streamId: screenStreamId,
    x: 640,
    y: 360,
    width: 1280,
    height: 720,
    zOrder: 1
}, {
    streamId: cameraStreamId,
    x: 160,
    y: 90,
    width: 320,
    height: 180,
    zOrder: 2
}]);
```

Step 6. Set the mirroring effect

Mirror the camera image, as the image collected by the camera is actually reversed.

```
videoEffectManager.setMirror({
    streamId: cameraStreamId,
    mirrorType: 1
});
```

Step 7. Add a watermark

Prepare an image object and add it to the video stream image as a watermark. Here, the watermark image is placed in the top-right corner.

```
var image = new Image();
image.src = './xxx.png'; // Note that the image address cannot be under another dom

videoEffectManager.setWatermark({
    image: image,
    x: 1230,
    y: 50,
    width: 100,
    height: 100,
    zOrder: 3
});
```

Step 8. Start stream push

Push the video stream with a PiP layout, mirrored image, and watermark output after the above steps to the server.

```
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=xxx');
```

Note

For more information on WebRTC-based stream mix APIs, see TXLivePusher.

On-Screen Comment and Chat

Last updated：2022-11-21 16:56:28

Overview

In the live streaming business, hosts and viewers usually need to interact in real time through various means such as on-screen comments and chat. However, integration of such features is complex. This document uses IM as an example to describe how to implement requirements such as on-screen comments, chat, and item recommendations during live streaming as well as possible problems and considerations, giving a glimpse of the live streaming business and requirements.

9:41
Joseph
ID : 123456
Oscar
Gave a flower
125K+
watch online
Welcome Daniel to the live room
Charlie: So cool!
Charlie: Awesome 😊
Welcome Daniel to the live room
Joseph: Welcome to my live room. Today let's enjoy more relaxing music.
Charlie: Good !
Damian: very good ! !
PK

Key Feature Description

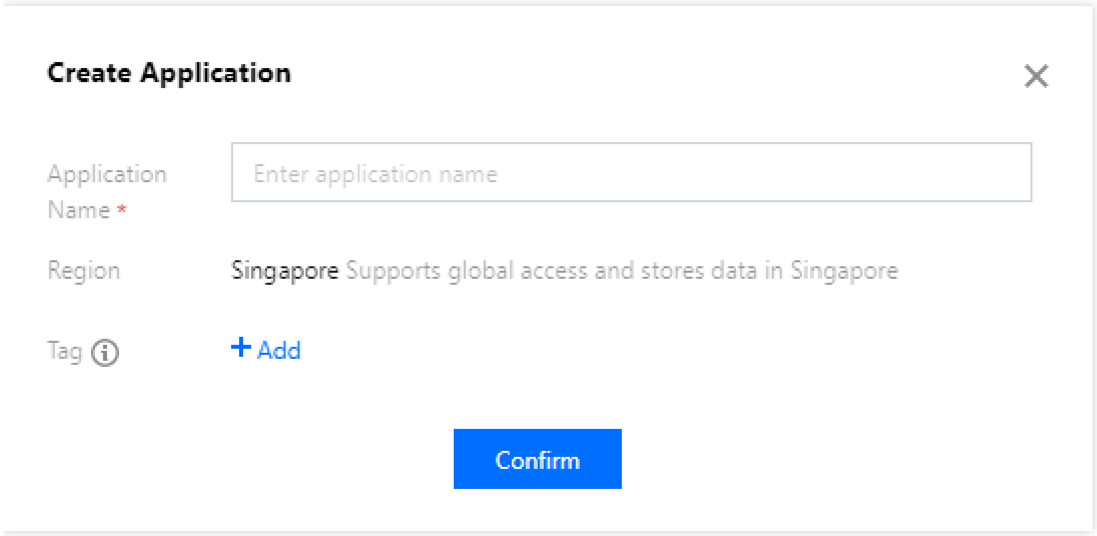
Feature	Description
Live on-screen comments, gifts, and likes	Hundreds of millions of concurrent messages can be sustained to build a friendly interaction experience.

Feature	Description
Various chat modes such as one-to-one and group live chat	Users can send messages and receive messages from other members in the same chat room. Diverse message types such as text, images, audio, and short videos can be pushed in real time, which encourages more user activity.
Item push in live shopping	In live shopping scenarios, when the host recommends an item, it needs to be immediately displayed in the item slot at the bottom of the screen and notified to all viewers. Notifications of new items are generally triggered by the virtual assistant.
Broadcast in a live room	The broadcast feature is similar to a system notice sent to live rooms. When the system admin delivers a broadcast message, all the live rooms under the `SDKAppID` will receive it.

Integration Method

Step 1. Create an application

To set up a live room in Tencent Cloud, you need to create an IM application in the console as shown below:


Create Application

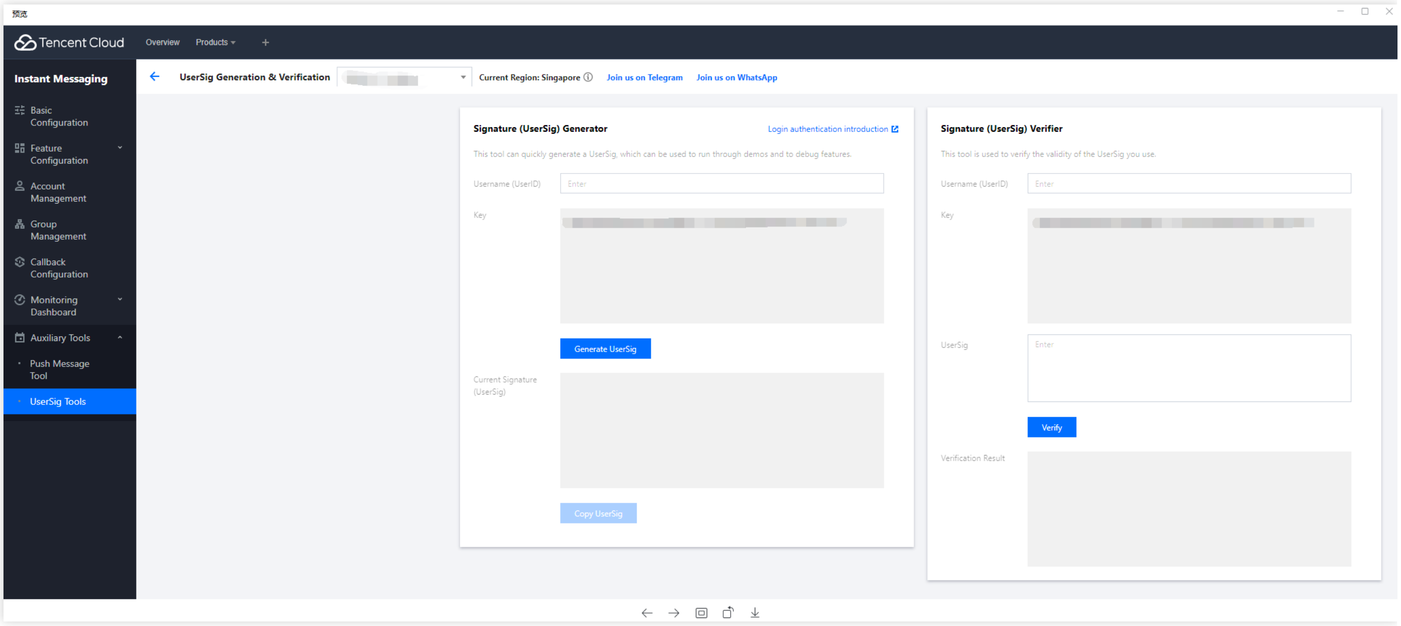
Application Name *	Enter application name
Region	Singapore Supports global access and stores data in Singapore
Tag	+ Add

Step 2. Complete the relevant configuration

The application created in step 1 is the Free edition, which applies only to development. In the production environment, you need to activate the Pro or Ultimate edition as needed. For more information on differences between different editions, see Pricing.

In live streaming scenarios, you need some extra configurations after creating the application.

- **Calculating the `UserSig` with a key**
  In the IM account system, the password required by a user login is calculated by the server with a key provided by IM. For more information, see Generating UserSig. In the development phase, to avoid holding back development on the client, you can also calculate the `UserSig` in the console as shown below:



Current Region: Singapore

Signature (UserSig) Generator

This tool can quickly generate a UserSig, which can be used to run through demos and to debug features.

Username (UserID): Enter

Key: [redacted]

Current Signature (UserSig):

Signature (UserSig) Verifier

This tool is used to verify the validity of the UserSig you use.

Username (UserID): Enter

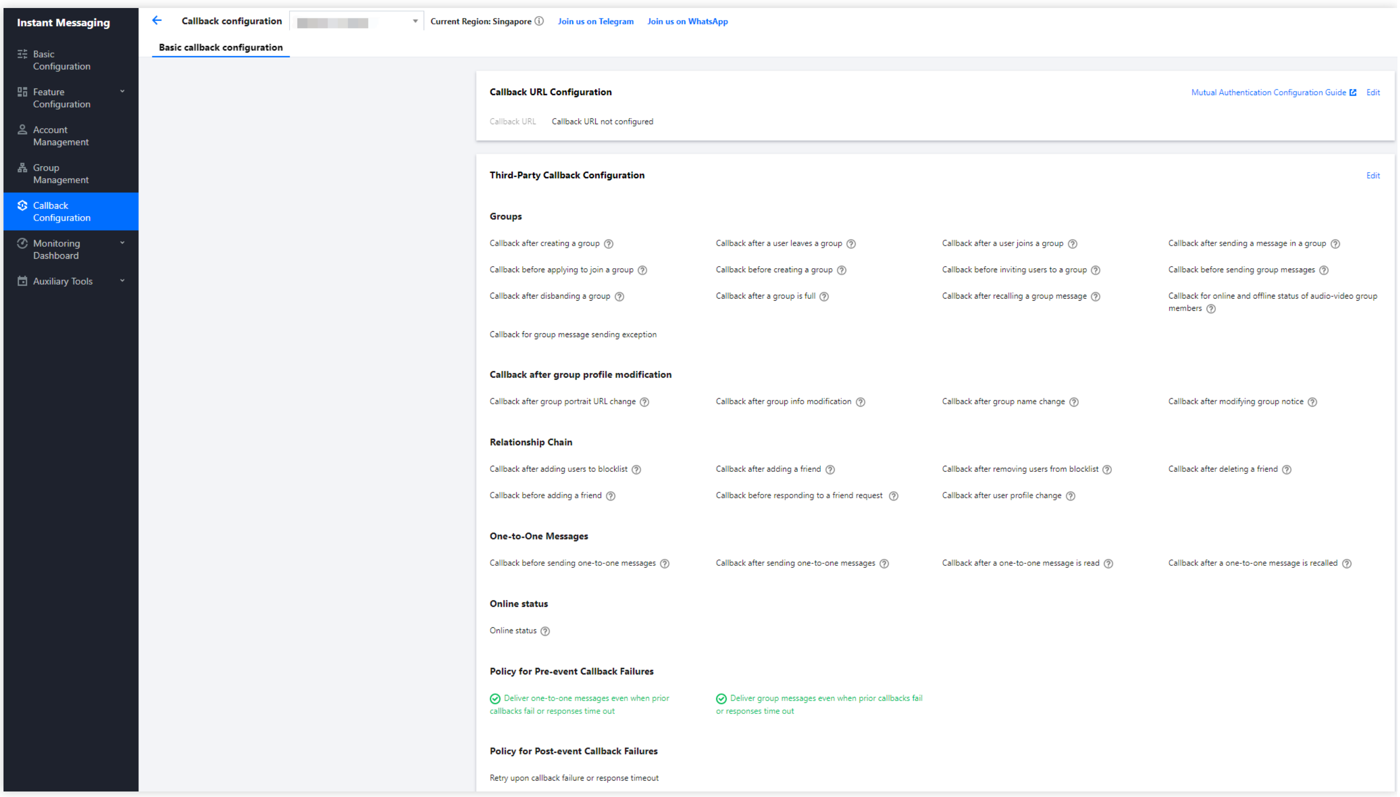
Key: [redacted]

UserSig: Enter

Verification Result:

- **Configuring an admin account**
  During live streaming, an admin may need to send messages to a live room and mute (remove) non-compliant users, which can be done through APIs on the IM server as described in RESTful API List. To call these APIs, you need to create an IM admin account. By default, IM provides an account with the `UserID` of `administrator`. You can also create multiple admin accounts as needed. Note that you can create up to five admin accounts.
- **Configuring the callback address and enabling the callback**
  To implement lucky draws based on on-screen comments, message statistics collection, sensitive content detection, and other requirements, you need to use the IM callback module, where the IM backend calls back the business backend in certain scenarios. You only need to provide an HTTP API and configure it in the **Callback**

configuration module in the console as shown below:



[redacted]

Callback URL Configuration

Callback URL Callback URL not configured

Third-Party Callback Configuration

Groups

Callback after creating a group	Callback after a user leaves a group	Callback after a user joins a group	Callback after sending a message in a group
Callback before applying to join a group	Callback before creating a group	Callback before inviting users to a group	Callback before sending group messages
Callback after disbanding a group	Callback after a group is full	Callback after recalling a group message	Callback for online and offline status of audio-video group members
Callback for group message sending exception			

Callback after group profile modification

Callback after group portrait URL change	Callback after group info modification	Callback after group name change	Callback after modifying group notice

Relationship Chain

Callback after adding users to blocklist	Callback after adding a friend	Callback after removing users from blocklist	Callback after deleting a friend
Callback before adding a friend	Callback before responding to a friend request	Callback after user profile change	

One-to-One Messages

Callback before sending one-to-one messages	Callback after sending one-to-one messages	Callback after a one-to-one message is read	Callback after a one-to-one message is recalled

Online status

Online status

Policy for Pre-event Callback Failures

✓ Deliver one-to-one messages even when prior callbacks fail or responses time out

✓ Deliver group messages even when prior callbacks fail or responses time out

Policy for Post-event Callback Failures

Retry upon callback failure or response timeout

Step 3. Integrate the client SDK

After completing the preparations, you need to integrate the IM and TRTC client SDKs into your project. You can select different integration options as needed. For detailed directions, see Get Started.

The following describes common features in a live room and provides best practices with implementation code.

Step 4. Develop key live room features

1. **Group type selection**

   The user chat section in live streaming scenarios has the following characteristics:

   - Users join and leave a group frequently, and group conversation information (unread count and `lastMessage`) doesn't need to be managed.
   - Users can join a group without approval.
   - Users send messages without caring about the chat history.
   - There are usually a large number of group members.
   - Group member information doesn't need to be stored.

Therefore, you can select **AVChatRoom** as the **Group type** for the live room based on the group characteristics of IM as described in Group System. IM audio-video groups (`AVChatRoom`) have the following characteristics:

- **Support interactive live streaming scenarios with unlimited group members.**

- Messages (group system notifications) can be pushed to all online members.
- Users can enter the group directly with no admin approval required.

> Note：
> The IM SDK for web allows users to join only one audio-video group (`AVChatRoom`) at a time. If a user logs in to an client and enters live room A, multi-client login is enabled in the console, and the user logs in to another client and enters live room B, the user will be removed from live room A.

2. **Configuration of on-screen comments, gifting, and likes for the live room**

- **On-screen comments**
  Audio-video groups (`AVChatRoom`) support on-screen comments, gifts, and like messages to build a friendly interaction experience.
  To create an on-screen comment, you can use the IM API to create a text or custom message. After the message is sent successfully, you need to get its text or custom attributes in the live room by receiving the OnRecvNewMessage() callback and then display it on the desired UI.
- **Gift**
  - Non-persistent connection requests from the client are sent to the business server, which involves the billing logic.
  - After fees are incurred, the sender can see that XXX sent the XXX gift (so that the sender can see the gift sent by himself/herself; when there are a large number of messages, the policy for discarding messages may be triggered).
  - After the fees are settled, you can call the server API to send the custom message (gift).
  - If multiple gifts are sent in a row, you need to merge the messages.
    - If the number of gifts is selected in advance, for example 99 gifts, you can send a message with `99` included in the parameter.
      - If gifts are sent multiple times and the total number is uncertain, you can send a message for every 20 gifts (the value can be adjusted) or clicks within a second. For example, if 99 gifts are clicked in a row, only five messages need to be sent after the optimization.
- **Like**
  - Unlike a gift message, a like message is not billed and directly sent on the client.
  - For like messages that need to be counted on the server, after traffic throttling is performed on the client, likes on the client are counted, and like messages within a short period of time are merged into one. The business server gets the like count in the callback before sending a message.
  - For like messages that don't need to be counted, the logic in step 2 is used, where the business server sends a message after traffic throttling is performed on the client and doesn't need to get the count in the callback before

sending a message.

3. **Item push in live shopping**
   When the host recommends an item, it needs to be immediately displayed in the item slot at the bottom of the screen and notified to all viewers. Notifications of new items are generally triggered by the virtual assistant. We recommend you implement notifications of new items by enabling the admin to modify a custom group field as follows:

   i. **Add a custom group field**

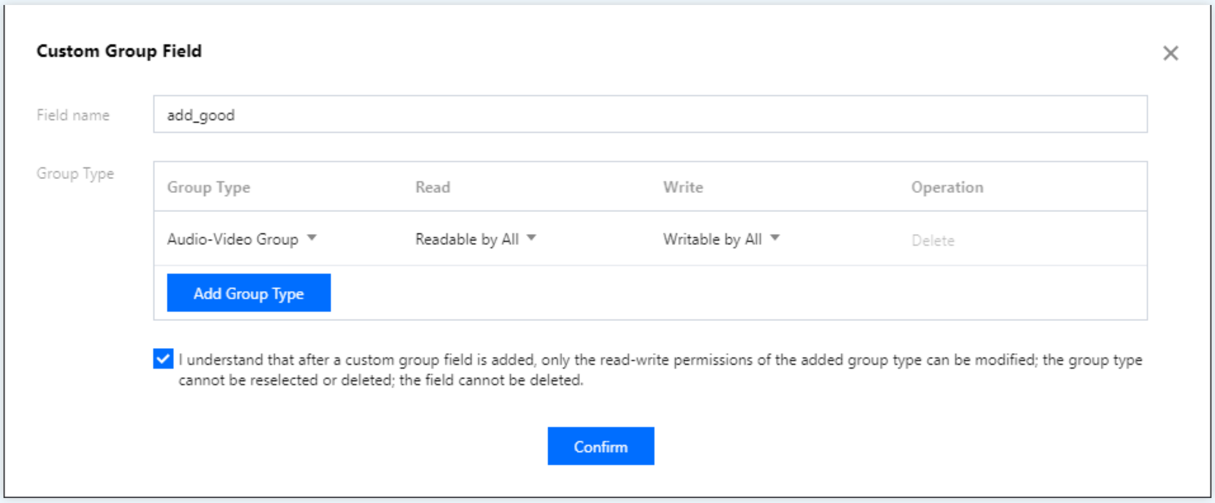
      a. Log in to the IM console, click the target IM application card, and select **Feature Configuration** > **Group configuration** on the left sidebar.

      b. On the **Custom Group Field** page, click **Add** in the top-right corner.

      c. In the pop-up dialog box, enter a field name and set the group types and read/write permissions.

      Note：

      - The field name can contain up to 16 characters, supporting letters, digits, and underscores (_). It cannot begin with a digit.
      - A custom group field and a custom group member field cannot have the same name.



Custom Group Field

Field name: add_good

Group Type

Group Type	Read	Write	Operation
Audio-Video Group	Readable by All	Writable by All	Delete

☑ I understand that after a custom group field is added, only the read-write permissions of the added group type can be modified; the group type cannot be reselected or deleted; the field cannot be deleted.

      d. Select **I understand that after a custom group field is added, only the read-write permissions of the added group type can be modified; the group type cannot be reselected or deleted; the field**

cannot be deleted. and click **Confirm**.

> Note：
> The custom group field will take effect in about ten minutes after configuration.

ii. **Use the custom group field**
The virtual assistant calls the RESTful API for modifying the profile of a group as the group admin at an appropriate time to update the custom group field, so as to send notifications of new items and notifications of live streaming status change in the live room.

4. **Broadcast in a live room**
The broadcast feature is similar to a system notice feature in the live room, but it differs from the latter in that it belongs to messaging. When the system admin delivers a broadcast message, all the live rooms under the `SDKAppID` will receive it.
The broadcast feature is currently available only for the Ultimate edition and needs to be enabled in the console. For more information on how to send a broadcast message on the business backend, see Broadcast Message of Audio-Video Group.

> Note：
> If you are not an Ultimate edition user, you can implement the feature by sending a custom group message on the server.

References

To implement more live room features such as user identity, user level, historical messages, and displaying the number of online users, see Live Room Setup Guide.