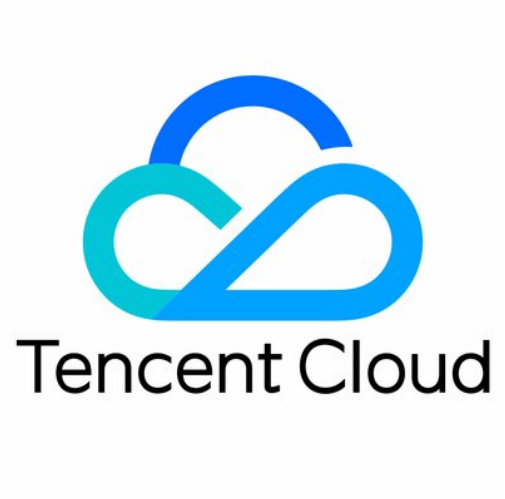
# Mobile Live Video Broadcasting

# FAQs

## Product Documentation



# Contents

# FAQs

# Basics

Last updated：2024-01-13 15:49:41

### What are publishing, live streaming, and video on demand?

**Publishing**: Stream publishing is the process of hosts publishing local video and audio to Tencent Video Cloud servers. It is sometimes called "RTMP publishing" as well.
**Live streaming**: In live streaming, video streams are generated in real time. It works only if someone is publishing live streams. A live streaming URL becomes invalid the moment the host stops publishing streams. Because live streams are played back in real time, audience cannot see a progress bar during live streaming.
**Video on demand**: In video on demand, it is the files in the cloud that are played back. Unless a file is deleted by its provider (e.g., Tencent Video), playback is possible at any time. Because videos are already stored in the server, audience can see a progress bar during playback.

### What are the requirements for a playback domain name in CSS?

The domain name can contain up to 29 characters and cannot contain uppercase letters. For more information, please see Adding Domain Name.

### Can I use the same domain name for playback and publishing? Can I use second-level domains?

You must use different domain names for playback and publishing, but you can distinguish them by second-level domains.
For example, you can use `123.abc.com` for publishing, and `456.abc.com` for playback.

### What publishing protocols can I use?

RTMP may not be a widely adopted protocol for live streaming, but it is the most common protocol used for stream publishing (publishing data from hosts to servers). Most video cloud services in the Chinese mainland use RTMP as their main publishing protocol. The MLVB SDK is also called RTMP SDK because its first feature module is stream publishing.

### What playback protocols can I use?

Common live streaming protocols include RTMP, HTTP-FLV, HLS, and WebRTC.
**RTMP** can be used for both publishing and playback. It works by splitting long video and audio chunks into short fragments and transmitting them as small data packets over the internet. RTMP supports encryption and therefore ensures privacy. However, the complicated splitting and reassembling processes add uncertainty to the stability of data transmission in high concurrency scenarios.

**HTTP-FLV** is developed by Adobe Systems and is a rather simple video format. It works by adding a header to large video and audio data chunks. This simplicity gives it a notable advantage in terms of latency control and high-concurrency performance. The only drawback is that HTTP-FLV is poorly supported on mobile browsers, but it is an ideal option for mobile apps.

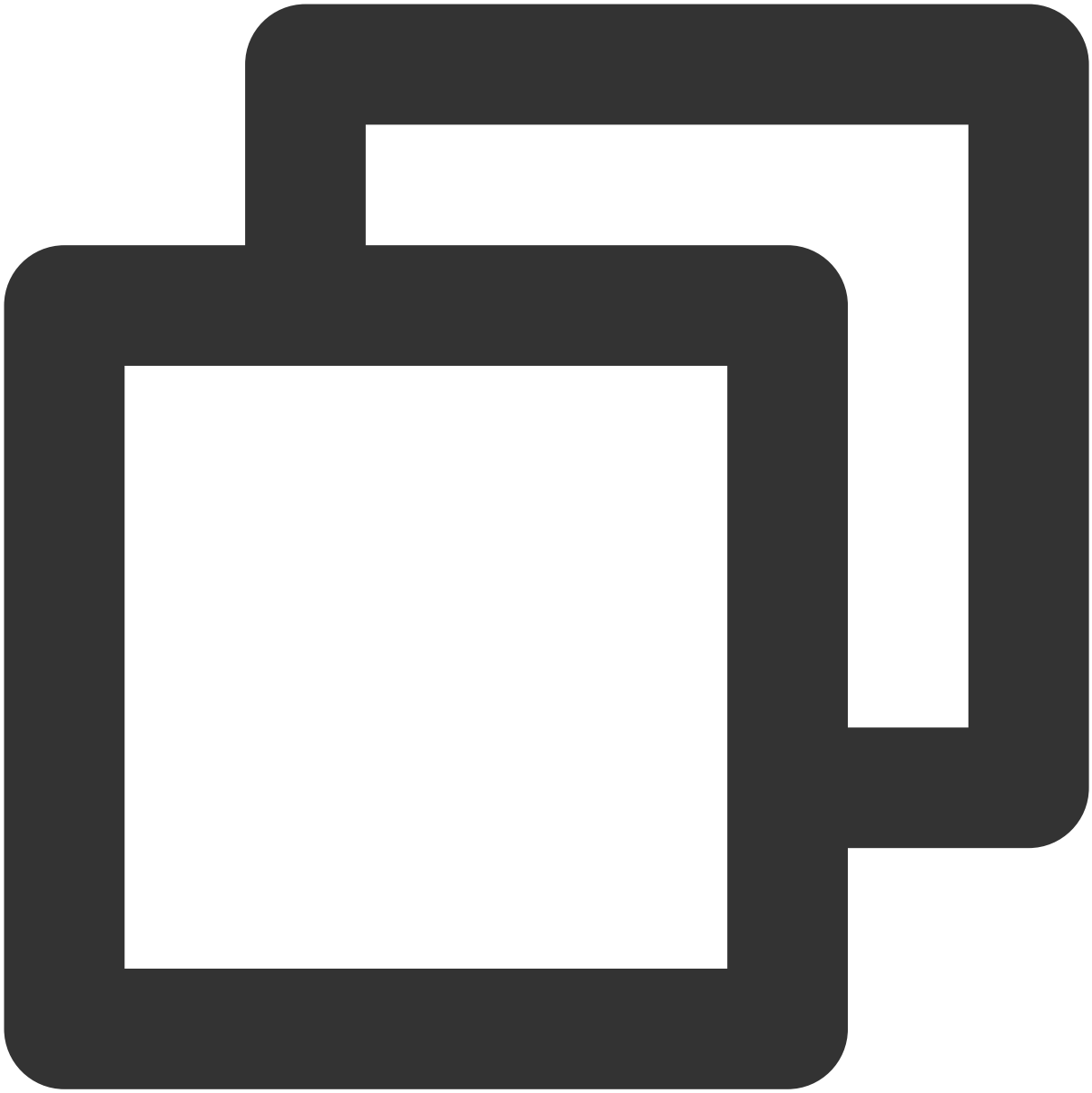
**HLS** is released by Apple. It breaks video streams into fragments of 5-10s and manages them using M3U8 playlists. The protocol ensures smooth playback as it is separate data chunks of 5-10s that the client downloads. However, it comes with high latency, normally about 10-30s. Unlike HTTP-FLV, HLS is well supported on iPhone and most Android browsers and is therefore often used for URL sharing on QQ and WeChat Moments.

**WebRTC** is short for Web Real-Time Communication. It is an API that allows real-time audio/video calls on web browsers. Supported by Google, Mozilla, and Opera, WebRTC specifications were published by the World Wide Web Consortium (W3C) on June 1, 2011. WebRTC is adopted by LEB, which is an ultra-low-latency version of LVB and offers streaming with millisecond latency. It is suitable for scenarios with high requirements on latency, such as online education, sports streaming, and online quizzes.

| Protocol | Pro | Con | Playback Latency |
|---|---|---|---|
| HTTP-FLV | Mature, suited for high-concurrency scenarios | SDK integration is required. | 2-3s |
| RTMP | Relatively low latency | Poor performance in high-concurrency scenarios | 1-3s |
| HLS (M3U8) | Well supported on mobile browsers | High latency | 10-30s |
| WebRTC | Lowest latency | SDK integration is required. | < 1s |

### What is the format of a playback URL?

A Tencent Cloud playback URL consists of a playback protocol prefix, domain name ( `domain` ), application name ( `AppName` ), stream name ( `StreamName` ), playback protocol suffix, authentication key, and other custom parameters. Below are examples:



```
rtmp://domain/AppName/StreamName?txSecret=Md5(key+StreamName+hex(time))&txTime=hex(
http://domain/AppName/StreamName.m3u8?txSecret=Md5(key+StreamName+hex(time))&txTime
http://domain/AppName/StreamName.flv?txSecret=Md5(key+StreamName+hex(time))&txTime=
https://domain/AppName/StreamName.m3u8?txSecret=Md5(key+StreamName+hex(time))&txTim
https://domain/AppName/StreamName.flv?txSecret=Md5(key+StreamName+hex(time))&txTime
webrtc://domain/AppName/StreamName?txSecret=Md5(key+StreamName+hex(time))&txTime=he
```

**Prefix**

RTMP: **rtmp://**

HTTP-FLV: **http://** or **https://**

HLS: **http://** or **https://**

WebRTC: **webrtc://**

**Application name ( `AppName` )**

Application name specifies the storage path of a live streaming file. It is **live** by default.

**Stream name ( `StreamName` )**

Stream name ( `StreamName` ) uniquely identifies a live stream.

**Authentication key and other custom parameters**

Authentication key: **txSecret=Md5(key+StreamName+hex(time))&txTime=hex(time)**.

### What are some common publishing methods?

**Camera on Android/iOS devices**: Third-party software or the MLVB SDK captures video data from the camera and publishes it to the publishing URL.

**Camera or screen recording tool on PCs**: Third-party software captures video data from the camera or records the screen and publishes the data to the publishing URL. Third-party publishing applications include OBS (recommended), XSplit, FMLE, etc.

**Video capturing device**: Connect an HD camcorder with HDMI or SDI output to an encoder and publish RTMP streams to live streaming applications. You need to set the RTMP publishing address of the encoder to your publishing URL.

If you use a webcam that supports RTMP, you can also set the RTMP publishing address of the webcam to your publishing URL.

**Converting video files to video streams**: Read video files and publish them as RTMP streams to your RTMP publishing URL. This can be achieved using FFmpeg commands, which works on Windows, Linux, and macOS.

### What are the differences between stream interruption and stream disabling?

**Stream interruption**: If a live stream is interrupted, publishing will stop, and audience will be unable to watch the stream. However, the host can resume publishing the stream.

**Stream disabling**: If a live stream is disabled, publishing will stop, and audience will be unable to watch the stream. The host cannot resume publishing the stream. You can disable a stream on the stream management page of the CSS console. Disabled streams can be found in the list of disabled streams. You can click **Enable** to enable a disabled stream.

### How do MLVB V1 APIs correspond to V2 APIs?

We offer a table that lists the relationships between V1 and V2 APIs. For details, please see Relationships Between MLVB SDK V1 and V2 APIs.

### Regarding privacy, what data (e.g., MAC address, IMEI) does the MLVB SDK collect?

The MLVB SDK complies with the privacy requirements of app stores. For details, please see Privacy Policy.

# Publish/Playback

Last updated：2024-01-13 15:49:41

## Is there an upper limit on the number of concurrent viewers?

CSS does not limit the number of concurrent viewers. A live stream can be viewed by as many users as the network allows. However, if you have set a bandwidth limit, new viewers will be unable to watch a live stream once the limit is reached.

## How do I use transcoding for playback?

You may want to use different bitrates and resolutions under different network conditions. You can create transcoding templates with different bitrates and resolutions in the console. For more information about transcoding, see Best Practice > CSS Encapsulating and Transcoding.

### Original definition, HD, and SD

The three commonly used playback bitrates for original-definition, HD, and SD streams are as follows:
For an original-definition stream, the playback bitrate is the same as the publishing bitrate.
For an HD (1080p) stream, the recommended playback bitrate is 2,000 Kbps.
For an SD (720p) stream, the recommended bitrate is 1,000 Kbps.

## How can I use time shifting for replay?

You can use the time shifting feature to replay highlights. The feature supports only the HLS protocol for the time being. For more information on time shifting and how to activate it, see Best Practices > CSS Time Shifting.

## How can I use HTTPS for playback?

If you want your playback domain name to support HTTPS, make sure you have a valid certificate and private key, and go to Domain Management, find your playback domain name, click **Manage**, select **Advanced Configuration**, and add configurations in **HTTPS Configuration**. It may take a while (2 hours) for the configurations to take effect, after which your streams can be played back over the HTTPS protocol.

## How can I use an acceleration node outside the Chinese mainland for playback?

CSS has CDN nodes across the Chinese mainland and around the world, with extensive coverage and high stability. If your end users are outside the Chinese mainland, you can select **Global Acceleration** or **Hong Kong/Macao/Taiwan (China Region) and other regions** for acceleration region when configuring your domain name in Domain Management.

**Note:**

Global acceleration supports only the HTTP-FLV and HLS protocols for the time being.

## How can I enable hotlink protection?

To prevent unauthorized users from accessing your playback URLs, which would consume your Tencent Cloud traffic, you are strongly advised to enable hotlink protection for your playback URLs to avoid potential losses caused by hotlinking. With CSS, hotlink protection for playback URLs is controlled by four parameters: `txTime`, `key` (hash key), `txSecret`, and the validity period.

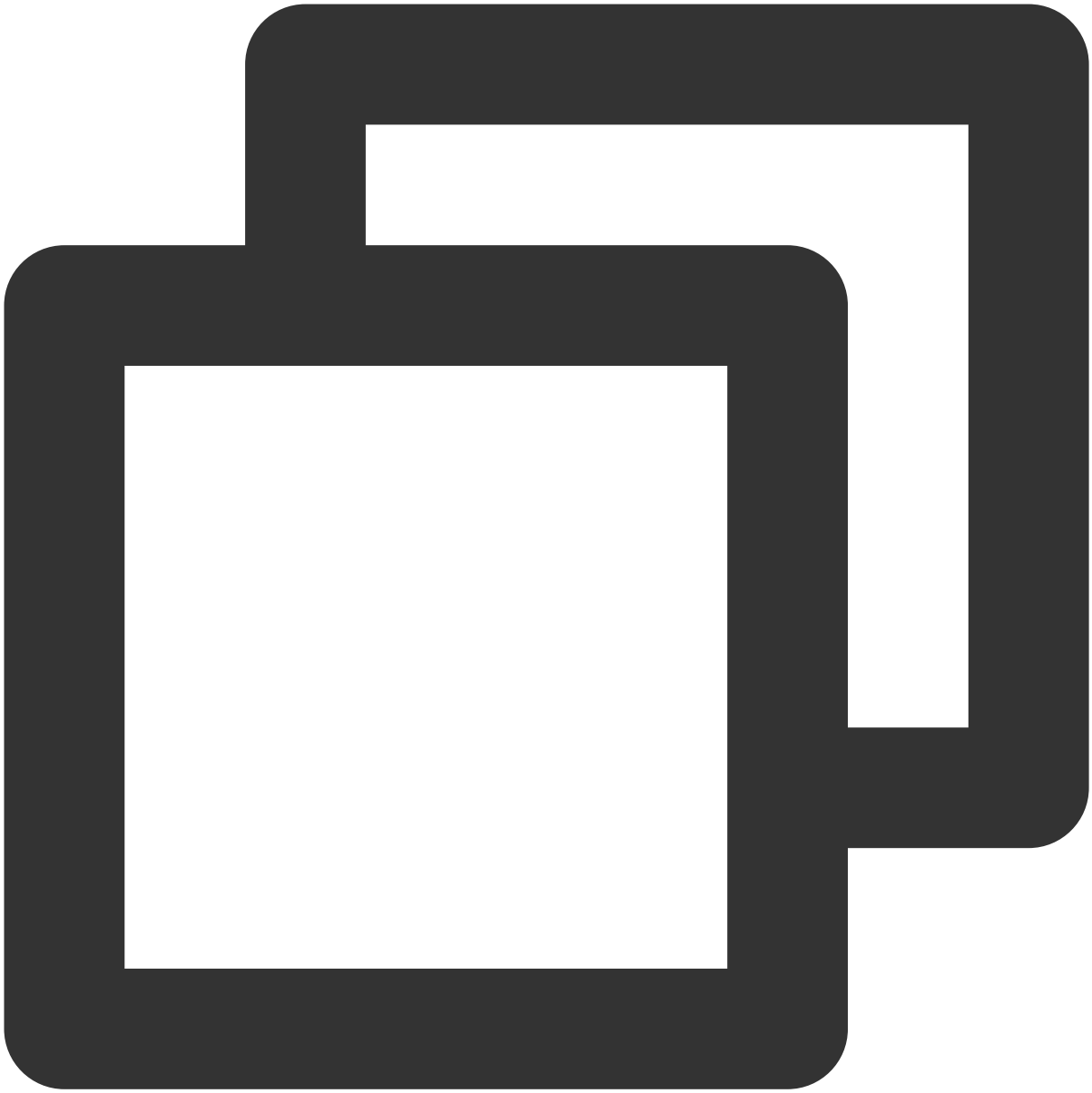
| Hotlink Protection Parameter | Description | Remarks |
| --- | --- | --- |
| txTime | Effective time of playback URL | It is a Unix hexadecimal time.<br>If the value of `txTime` is greater than the time of a request, the stream can be played back successfully; otherwise the request will be rejected. |
| key | MD5 key | You can customize a key as well as set a master and slave key.<br>If your master key is leaked, you can use the slave key to splice playback URLs and change the value of the master key. |
| txSecret | Encryption parameter in playback URL | The value of this parameter is calculated based on `key`, `StreamName`, and `txTime` using the MD5 algorithm.<br>`txSecret` = MD5 (key+StreamName+txTime) |
| Validity period | Validity period of playback URL | It must be greater than 0.<br>If `txTime` is set to the current time and the validity period is 300 seconds, then the playback URL expiration time is the current time + 300 seconds. |

### Hotlink protection URL calculation

The calculation of a hotlink protection URL requires three parameters: `key` (a random string), `StreamName` (stream name), and `txTime` (in the hexadecimal format).

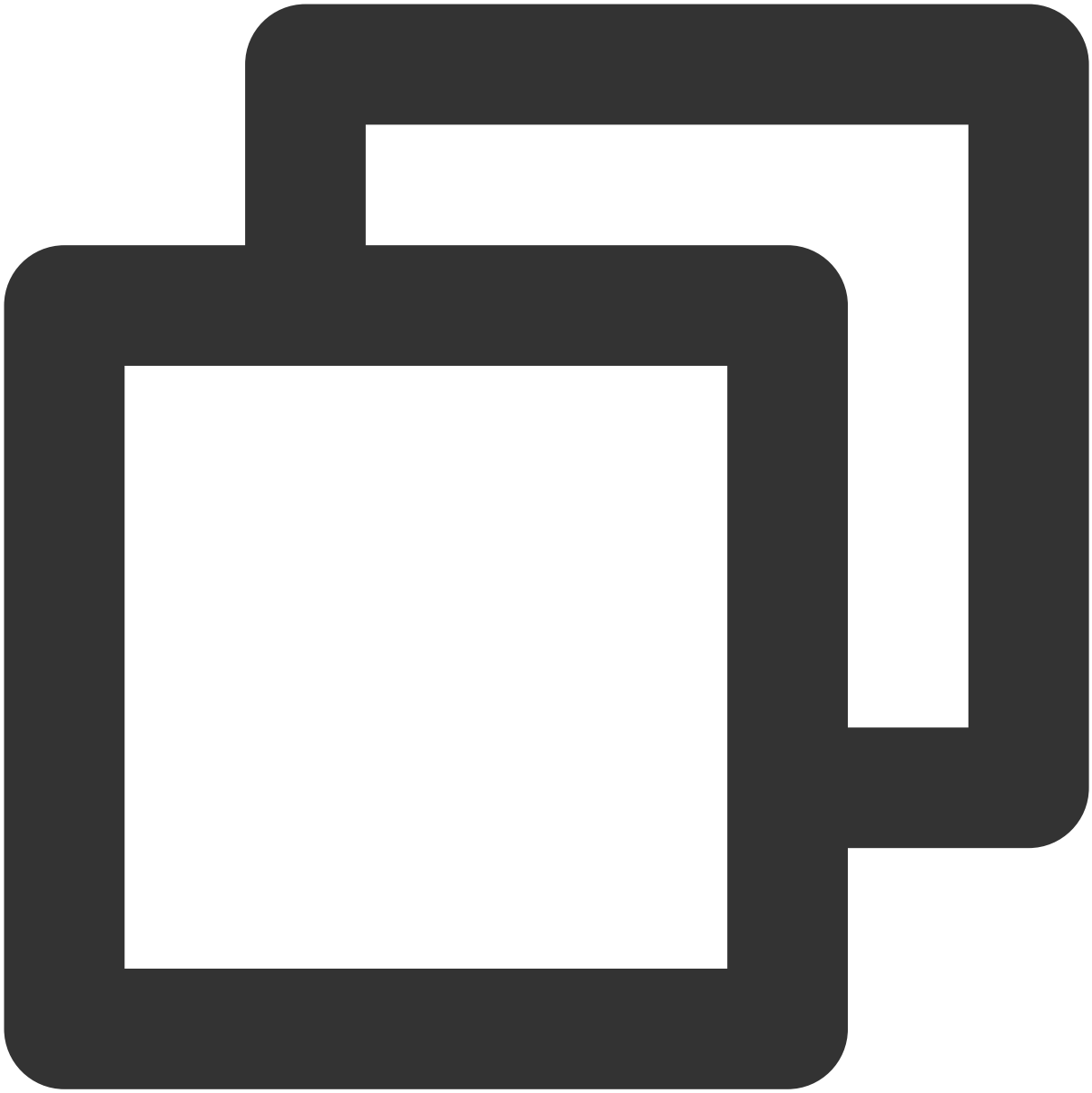
Suppose you set `key` to **somestring**, the stream name (`StreamName`) to **test**, `txTime` to **5c2acacc** (2019-01-01 10:05:00), the HD bitrate to **900 Kbps**, and the name of the transcoding template to **900**.

The playback URL for an original-definition stream would be:



```
txSecret = MD5(somestringtest5c2acacc) = b77e812107e1d8b8f247885a46e1bd34
http://domain/live/test.flv?txTime=5c2acacc&txSecret=b77e812107e1d8b8f247885a46e1bd
http://domain/live/test.m3u8?txTime=5c2acacc&txSecret=b77e812107e1d8b8f247885a46e1b
```

The playback URL for an HD stream would be:



```
txSecret = MD5(somestringtest_9005c2acacc) = 4beae959b16c77da6a65c7edda1dfefe
http://domain/live/test_900.flv?txTime=5c2acacc&txSecret=4beae959b16c77da6a65c7edda
http://domain/live/test_900.m3u8?txTime=5c2acacc&txSecret=4beae959b16c77da6a65c7edd
```

**Enabling hotlink protection**

1. Log in to the CSS console and click **Domain Management**.
2. Find and click your playback domain name or click **Manage** to enter the details page.
3. Select **Access Control** and click **Edit**.
4. Enable **Playback Authentication** and click **Save**.

**Note:**

It takes **30 minutes** for the configuration to take effect.

HTTP-FLV: ongoing playback can continue even after the URL expires, but new requests for playback via the URL will be rejected.

HLS: as HLS breaks a stream into short chunks, it keeps requesting M3U8 files to get the latest TS segments. Suppose you set `txTime` to the current time and the validity period to 10 minutes, then an HLS playback URL request sent 10 minutes after the current time will be rejected. To avoid this, you can update the HLS request URL dynamically on the server or set longer validity periods.

### What format requirements must a master key for playback authentication meet? Is there any limit on its validity period?

A master key for playback authentication can be a random combination of uppercase and lowercase letters and digits and must not be longer than 256 bits.

We recommend that you set the validity period to the duration of a live stream.

### I recorded live streams. How can I get the recording files?

Recording files are automatically saved in the VOD system after generation and can be found via:

VOD console

Recording event notification

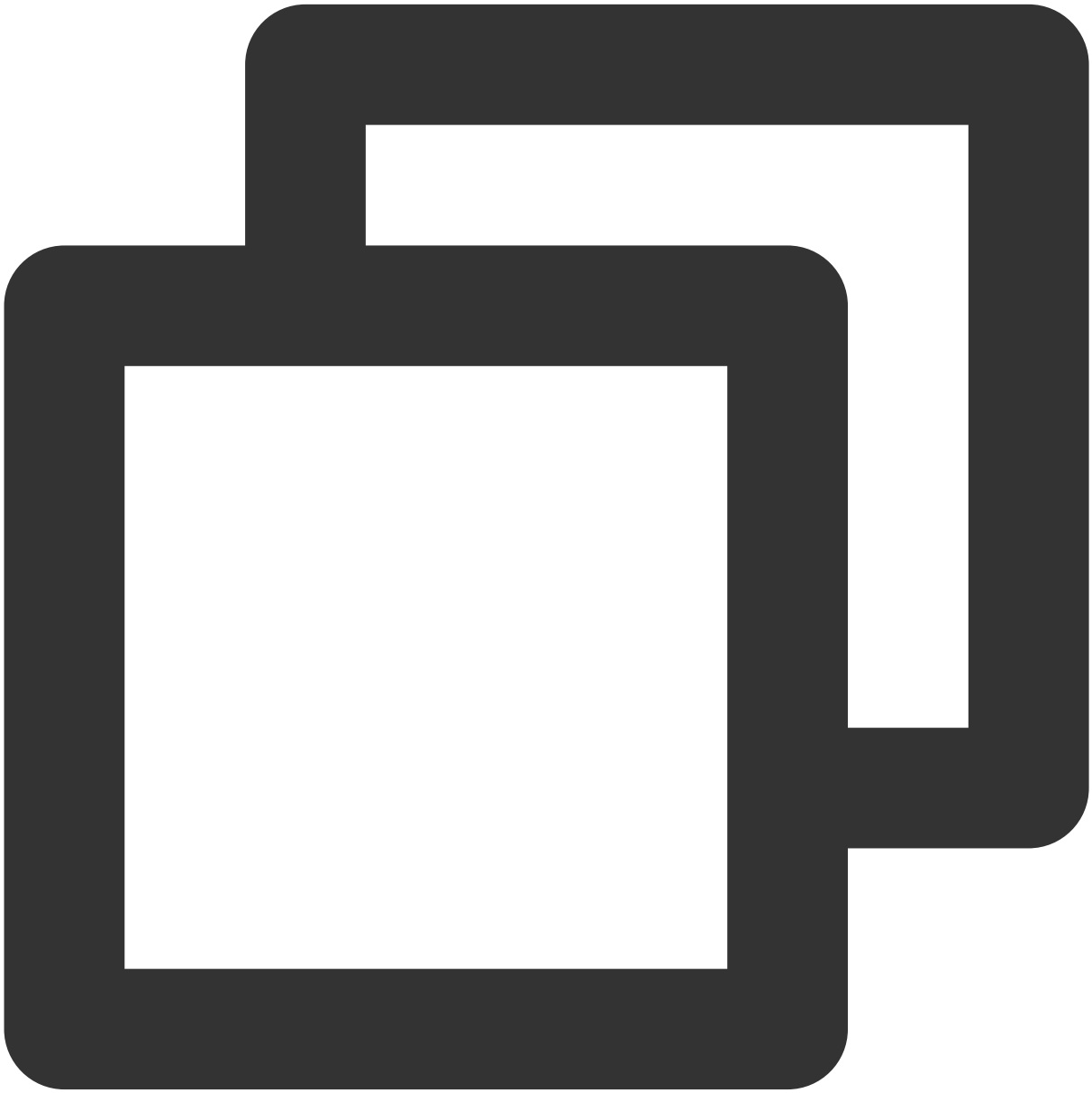
VOD querying APIs

# License

Last updated：2024-01-13 15:49:41

### How to obtain the package name for an Android project?

You can obtain the package name in the `Mainfest.xml` file of your Android project.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.huawei.player"
```

```
    android:versionCode="20181111"
    android:versionName="1.0">
```

### How to obtain the bundle ID for an iOS project?

You can obtain the bundle ID in **General** > **Identity** in Xcode, as shown below:


### Can I renew a trial license after it expires?

You can use a trial license for 28 days at most. When it first expires after 14 days, you can renew it for another 14 days. After 28 days, please purchase a license.

If you renew a trial license within the first 14 days, the license will expire 28 days after the time of license application; if you renew a trial license that has expired once, the renewed license will expire 14 days after renewal.

For example, if you apply for a trial license at `2021-08-12 10:28:41`, it will expire 14 days later, at `2021-08-26 10:28:41`.

You can renew the trial license once for free. If you renew it within the first 14 days, it will expire at `2021-09-09 10:28:41`; if you renew it after the first 14 days, at `2021-08-30 22:26:20`, it will expire at `2021-09-13 22:26:20`.

### Can I change the package name for an Android project or the bundle ID for an iOS project if I use a trial license?

Yes, you can.

In the CSS console, go to **MLVB SDK** > **License Management** and click **Edit** to change the bundle ID and package name.

### Can I change the package name for an Android project or the bundle ID for an iOS project if I use an official license?

No, you can't.

### Can I use a license for multiple applications at the same time?

Each license can be bound to only 1 package name and bundle ID. If you want to use MLVB features in multiple applications, you need to purchase multiple licenses.

### Do I have to purchase an MLVB license?

You can use the stream publishing feature of the MLVB SDK only if you have an MLVB license.

**Note:**

You cannot unlock MLVB features with a UGSV license.

### Is there a self-help purchase page for MLVB licenses?

No, there isn't.

With a live publishing license (previously LiteAV_Smart license), you can use the LiteAV_Smart SDK on iOS and Android. With an enterprise edition license, you can use MLVB Enterprise Edition on iOS and Android. To purchase the licenses, please contact our sales rep.

### How does a live publishing license (previously LiteAV_Smart license) differ from an enterprise edition license?

You can use a live publishing license (previously LiteAV_Smart license) to unlock the publishing and playback features of the SDK, as well as basic beauty filters such as skin brightening and skin smoothing.

An enterprise edition license gives you access to additional features including advanced beauty filters (e.g. eye enlarging and face slimming), green screen, animated stickers, AI keying, etc. You can also use makeup and gesture materials to implement more features.

**Note:**

You can use a live publishing license (previously LiteAV_Smart license) to unlock the publishing and playback features in all three editions of the MLVB SDK.

The advanced beauty filters provided by MLVB Enterprise Edition must be unlocked with an enterprise edition license.

### Can I use multiple licenses under the same account?

There is no limit on the number of licenses you use for an account, but to better manage your resources, you are advised to renew an existing license to extend your access to the SDK instead of adding a new license with the same package name.

### Can I add multiple licenses with the same package name?

Yes, you can. The validity periods of different licenses are calculated separately. You are not advised to add multiple licenses with the same package name.

### Can I modify a license?

You can renew an MLVB license to extend its validity period, but you cannot modify the package name of a license. Before adding a license, please make sure that your package name is not already used by another app in Google Play

Store.

## I added multiple licenses. Why do they have the same `licenseurl` and `key`?

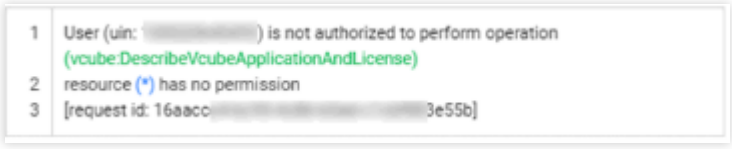
By default, licenses under the same account are assigned the same `licenseurl` and `key`. This ensures that trial licenses, official licenses, and licenses with different package names can share the API information.

**Note:**

You are not advised to commercially launch an app that uses a trial license. You can upgrade to the official version simply by adding an official license. You don't have to modify the `licenseurl` or `key` in APIs.

## I have granted a sub-account full access to CSS and VOD, but why can't it access the license page of the console?

**Screenshot:**




**Analysis:**

We updated APIs along with the MLVB SDK licenses. To access the license page of the console with a sub-account, you must use your root account to grant it separate access to licenses.

If you want to allow your sub-account to query licenses only, associate the QcloudVCUBEReadOnlyAccess policy.

If you want to give your sub-account full access to licenses, associate the QcloudVCUBEFullAccess policy.

For more information on how to associate permission policies with users/user groups, please see Authorization Management.

**Note:**

All license operations are now independent of CSS and VOD. That means the QcloudVODFullAccess and QcloudLIVEFullAccess policies no longer apply to license APIs. You must grant access to licenses separately as described above.

# Publishing/Playback URL

Last updated：2024-01-13 15:49:41

## Prerequisites

You have signed up for a Tencent Cloud account and activated the CSS service.
You have applied for a domain name at Tencent Cloud Domain Service.
You have added publishing/playback domain names in the CSS console > **Domain Management**. For detailed directions, please see Adding Domain Name.
You have configured CNAME for your domain names.

## Generating live streaming URLs in the console

1. Log in to the CSS console.
2. Go to **CSS Toolkit** > **Address Generator**, and do the following:
1. Select a domain type.
2. Select the domain name you have added in **Domain Management**.
3. Enter a custom `AppName` value ( `live` by default). `AppName` is used to differentiate the paths of applications under the same domain name.
4. Enter a custom `StreamName` value.
5. Select an expiration time for the address.
6. Click **Generate Address** to generate your publishing/playback address.

**Note:**

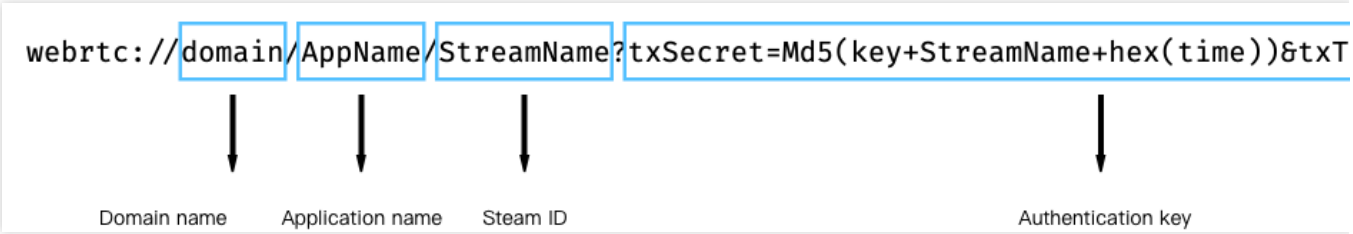
`AppName` is a custom value and can contain only letters, numbers, and special characters.
Here is another way to generate publishing addresses: In **Domain Management**, find the publishing domain name you want use to generate a publishing address, click **Manage**, select **Push Configuration**, enter an expiration time for the address and a custom `StreamName value` , and click **Generate Push Address**.

## Viewing the sample code of publishing URLs

Go to **Domain Management** of the CSS console, find the publishing domain name you created, click **Manage**, and select **Push Configuration**. Scroll down and you will find **Push Address Sample Code** (for PHP and Java). The code demonstrates how to generate a hotlink protection address. For detailed directions, see Push Configuration.

## Splicing publishing URLs

If you run a large number of live streaming rooms, it is impossible to manually generate a publishing and playback URL for each host. In such cases, you can use the server to **automatically splice** the URLs. Any URL that meets Tencent Cloud standards can be used for publishing. A standard publishing URL consists of four parts, as shown below:




**Domain**

Domain name for publishing, which can be the default publishing domain name provided by Tencent Cloud CSS or a publishing domain name that you have added and created a CNAME record for

**AppName**

Application name, which is a custom value and `live` by default

**StreamName (stream ID)**

Custom stream name, which is the unique ID of a live stream. We recommend that you use a random numeric or alphanumeric string for this parameter.

**Authentication key (optional)**

An authentication key consists of `txSecret` and `txTime` : `txSecret=Md5(key+StreamName+hex(time))&txTime=hex(time)` .

If publishing authentication is enabled, the URL used for publishing must contain an authentication key. If publishing authentication is disabled, the publishing URL ends before “?”.

**txTime (URL expiration time)**

The time when the URL expires, in the format of hexadecimal Unix timestamp

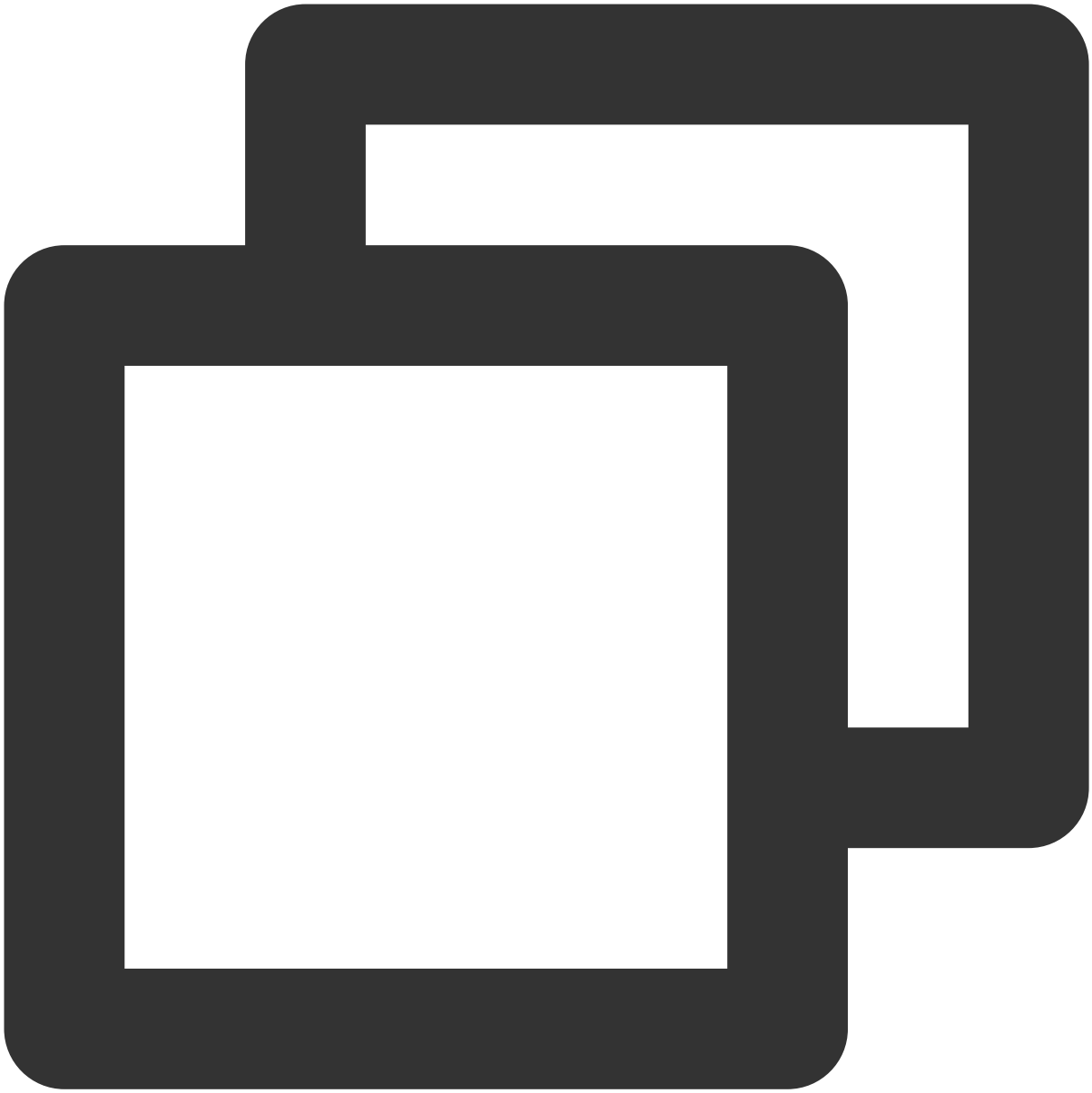
**Note:**

For example, `5867D600` means that the URL expires at 00:00:00, January 1, 2017. The validity period should neither be too short nor too long. Most of our clients set `txTime` to a point 24 hours or longer from the current time. If the validity period is too short, if a host is disconnected during a live stream, he or she may be unable to resume publishing due to the expiration of the publishing URL.

**txSecret (hotlink protection signature)** `txSecret` can prevent attackers from forging your backend to generate publishing URLs. For the calculation method, see Best Practice > Hotlink Protection URL Calculation.

## Splicing playback URLs

A playback URL consists of a playback protocol prefix, domain name ( `domain` ), application name ( `AppName` ), stream name ( `StreamName` ), playback protocol suffix, authentication key, and other custom parameters. Below are a few examples.



```
webrtc://domain/AppName/StreamName?txSecret=Md5(key+StreamName+hex(time))&amp;txTim
http://domain/AppName/StreamName.flv?txSecret=Md5(key+StreamName+hex(time))&amp;txT
rtmp://domain/AppName/StreamName?txSecret=Md5(key+StreamName+hex(time))&amp;txTime=
http://domain/AppName/StreamName.m3u8?txSecret=Md5(key+StreamName+hex(time))&amp;tx
```

**Prefix**

| Playback Protocol | Playback Prefix | Notes |
| --- | --- | --- |
| WebRTC | webrtc:// | Strongly recommended; best instant streaming performance; ultra-high concurrency |

| | | |
|---|---|---|
| HTTP-FLV | http:// or https:// | Recommended, excellent instant streaming performance; high concurrency |
| RTMP | rtmp:// | Not recommend; poor instant streaming performance; unable to handle high concurrency |
| HLS (M3U8) | http:// or https:// | We recommend HLS for mobile clients and for the Safari browser on macOS. |

**Domain**Domain name for playback, which must be a domain you have added and created a CNAME record for

**AppName**

Application name, which is a custom value ( `live` by default) that identifies the storage path of a live streaming media file

**StreamName (stream name)**

Custom stream name, which is the unique ID of a live stream. We recommend that you use a random numeric or alphanumeric string for this parameter.

**Authentication key (optional)**

An authentication key consists of `txSecret` and `txTime` :

`txSecret=Md5(key+StreamName+hex(time))&txTime=hex(time)` .

If playback authentication is enabled, the URL used for playback must contain an authentication key. If it is disabled, the playback URL does not need to contain "?" and the content following it.

**txTime (address expiration time):** the time when the URL expires, in the format of hexadecimal Unix timestamp

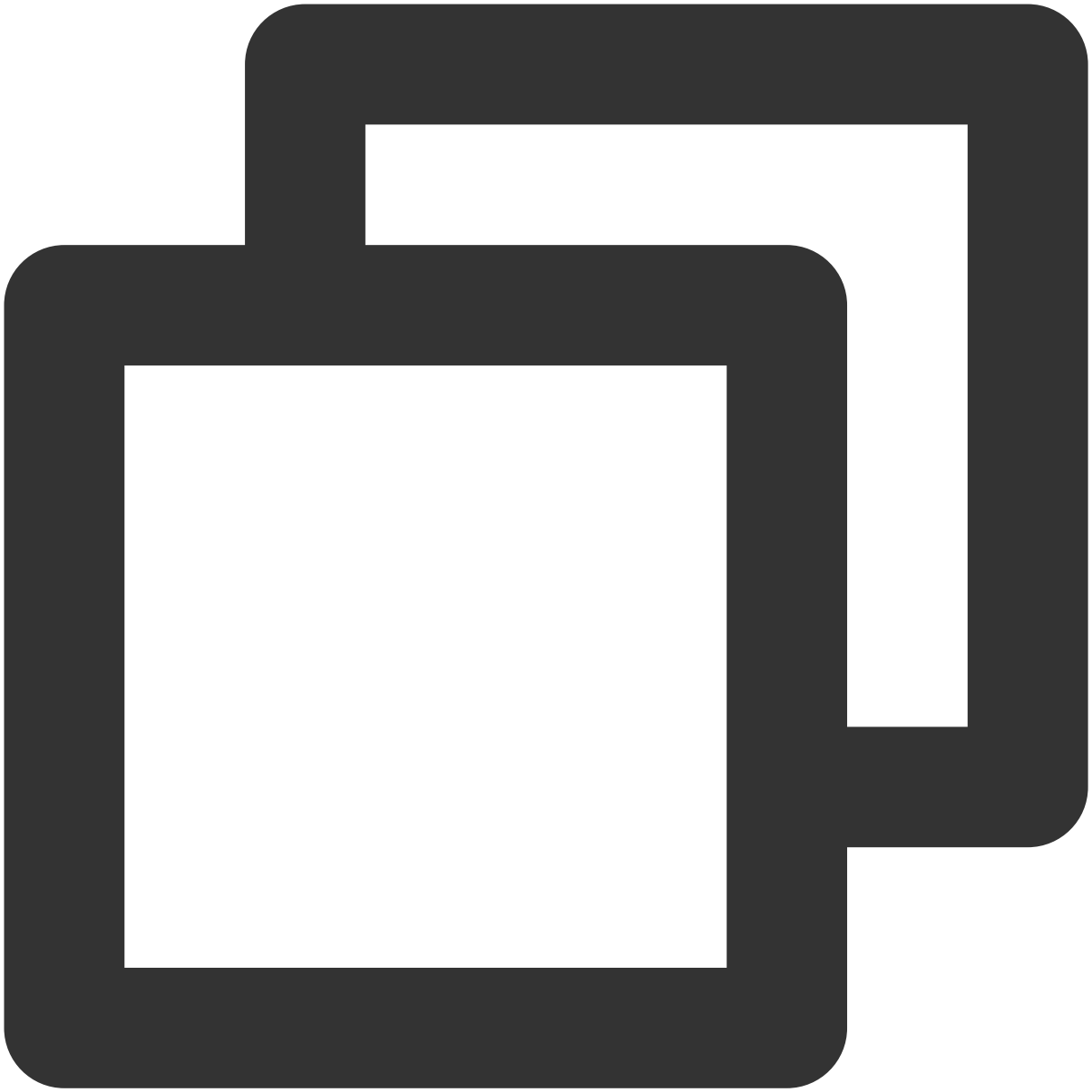
**txSecret (hotlink protection signature):** a signature that prevents attackers from forging your backend to generate playback URLs. For the calculation method, see Best Practice > Hotlink Protection URL Calculation.

## Splicing URLs for RTC-based mic connect/host competition

You need to splice publishing and playback URLs for RTC-based mic connect and host competition.

### Publishing URL

You need to splice a publishing URL by yourself in your project code. The format is as shown below.



```
trtc://cloud.tencent.com/push/streamid?sdkappid=1400188888&userId=A&usersig=xxxxx
```

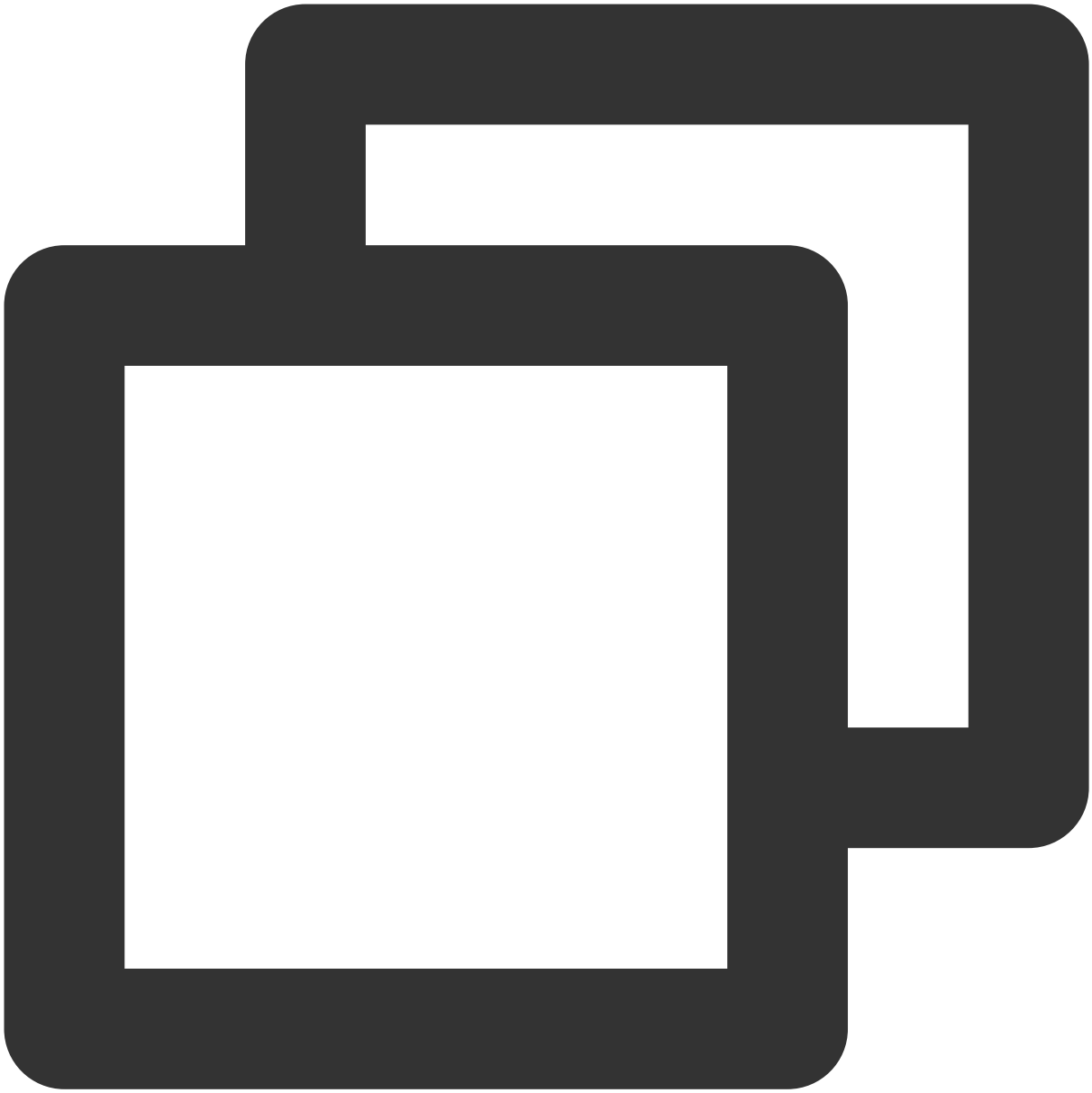
The table below lists the key fields in a publishing URL and their meanings.

| Field | Description |
| --- | --- |
| trtc:// | Prefix of a publishing URL for interactive live streaming |
| cloud.tencent.com | Dedicated domain name for interactive live streaming, which you must not modify |
| push | Identifier, which indicates publishing |

| | |
|---|---|
| streamid | Stream ID, which is a custom value specified by yourself |
| sdkappid | The `SDKAppID` generated in “Activate TRTC” |
| userId | User ID of the host, which is a custom value specified by yourself |
| usersig | User signature calculated from the key obtained in “Activate TRTC” |

**Playback URL**

You need to splice a playback URL by yourself in your project code. The format is as shown below.



```
trtc://cloud.tencent.com/play/streamid?sdkappid=1400188888&userId=A&usersig=xxxxx
```

The table below lists the key fields in a playback URL and their meanings.

| Field | Description |
| --- | --- |
| trtc:// | Prefix of a playback URL for interactive live streaming |
| cloud.tencent.com | Dedicated domain name for interactive live streaming, which you must not modify |
| play | Identifier, which indicates playback |

| | |
|---|---|
| streamid | Stream ID, which is a custom value specified by yourself |
| sdkappid | The `SDKAppID` generated in “Activate TRTC” |
| userId | User ID of the host, which is a custom value specified by yourself |
| usersig | User signature calculated from the key obtained in “Activate TRTC” |

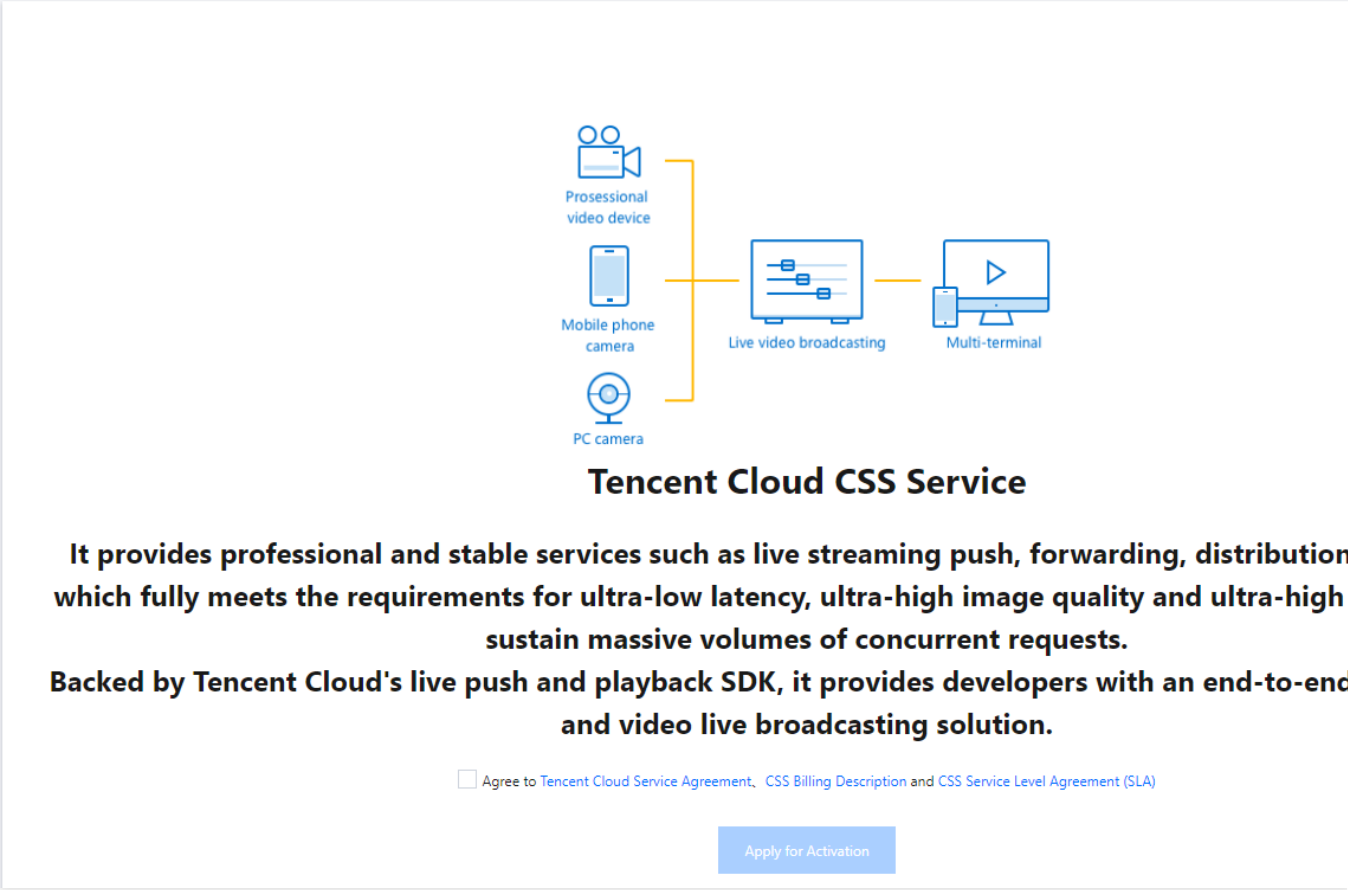
# Other Tencent Cloud Services

Last updated：2024-01-13 15:49:41

## Cloud Streaming Services (CSS)

### How to activate CSS?

Go to the CSS console, on the service activation page, read the agreement, check **Agree**, and click **Apply for Activation**.




### How do I enable hotlink protection keys?

Hotlink protection keys are a security mechanism that ensures only your app users can publish streams. You can change your hotlink protection settings anytime on the Domain Management page of the CSS console. For details, please see Playback Authentication Settings.

### How to obtain authentication keys for API calls?

For your backend server to call cloud APIs of CSS, it will need authentication keys to verify the legitimacy of the calls. For details on how to obtain authentication keys for API calls, please see Signature v3.

### How to configure a URL for event callbacks?

Tencent Cloud allows you to register callbacks for some live streaming events. It sends notifications to the URL you specify using the HTTP POST method. For how to configure a URL for event callbacks, please see CSS Callback.

## Video on Demand (VOD)

### How to activate VOD?

Go to the VOD console to activate VOD. The service is charged daily by default.

### How do I query my VOD `APPID` ?

Each Tencent Cloud account is assigned a unique `APPID` , which you can view in Account Information of the Account Center.

## Instant Messaging (IM)

### How to activate IM?

Go to the IM console.

The IM application list of a newly verified account is empty. To create an application, click **Create Application**, fill in the information, and click **Confirm**.

**Note:**

You can also create an IM application in the CSS console. Log in to the CSS console, go to **MLVB SDK** > **Application Management**, click **Create Application**, enter an application name and description, and click **Confirm**.

### What is `SDKAppID` ?

`SDKAppID` is a number, that represents a product under your account. If you have multiple products, each product will have a `SDKAPPID` .

### What is administrator?

IM provides a series of RESTful APIs that allows your backend server to use IM features directly, for example, to create a group, send system messages, and remove a user from a group. However, the RESTful APIs can only be called by an administrator. That is to say, you need a username that is `Administrator` and its corresponding password ( `UserSig` ). For more information, please see RESTful API Overview.

**Note:**

`UserSig` is the password needed to log in to IM. For details on how to obtain it, please see Obtaining a Key.

# Cloud Object Storage (COS)

### How to activate COS?

All Tencent Cloud accounts can use the COS service after verification. Go to the COS console, click **Bucket List** > **Create Bucket**, fill in the information, and click **Create** to create a bucket.

### What is bucket?

Put simply, buckets function as **disk partitions**. When you pay for Tencent Cloud's COS service, it's like purchasing a hard disk. It's a common practice to partition and format a disk before using it to store data. When you create a bucket in COS, it's like creating a hard disk partition.

### How do I query `BucketName` ?

`BucketName` is the name you give to a bucket during creation.

### How do I query `AppID` , `SecretId` , or `SecretKey` ?

To view the information, on the key management page of the COS console, click **Cloud API Key**.
In COS, `APPID` is bound to `SecretId` and `SecretKey` . They are required to call COS APIs. COS is a cloud service with high security requirements, so unless a correct key is passed in, an API request will be rejected by Tencent Cloud.

# Cloud Virtual Machine (CVM, optional)

You can use your own server to deploy backend scripts for your project, but we recommend Tencent Cloud CVM for greater expertise and reliability. Please note that Tencent Cloud's distributed cloud databases can only be accessed via Tencent Cloud's CVMs.
Log in to the CVM console, select **Instances** on the left sidebar, and click **Create** to go to the CVM purchase page.
**Note:**
For CVM image, we suggest that you select a Linux image with Nginx + PHP + MySQL from the image market. Complete the subsequent steps as prompted. The CVM becomes available after the image is installed.

# TencentDB for MySQL (optional)

### How to activate TencentDB for MySQL?

Please see Purchase Methods.

### How to use a database?

Please refer to:

Initializing MySQL Instance

Connecting to MySQL Instance

Instance Management Page

# Latency

Last updated：2024-01-13 15:49:41

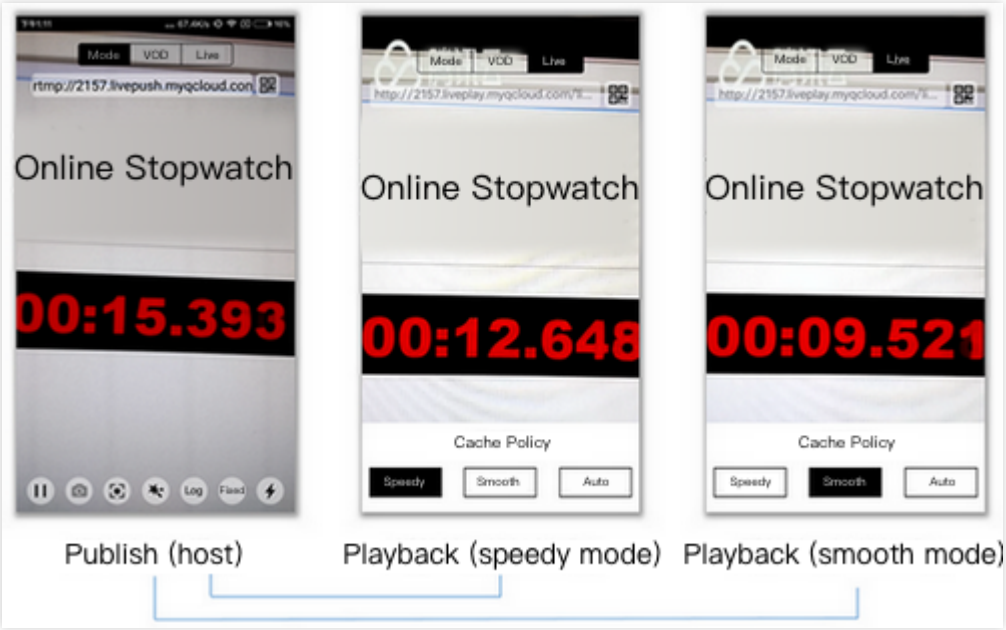
If you publish streams via RTMP and play streams via HTTP-FLV, the latency is generally about 2-3 seconds. If you experience high latency, follow the steps below to troubleshoot the problem.

### Step 1. Check your playback protocol

The latency tends to be high if you use HLS (M3U8) for playback. HLS is a streaming protocol developed by Apple. It works by breaking streams into (usually 3 or 4) TS segments of 5 seconds or longer, which results in an overall latency of 10-30 seconds.
Therefore, if you have to use HLS (M3U8) for playback, you can reduce latency by cutting the number of segments or the length of each segment, but this may increase stuttering. You can submit a ticket or contact our technical support engineers for help.

### Step 2. Check player settings

The player of the MLVB SDK supports three latency control modes: Speedy, Smooth, and Auto. For more information about their settings, please see Latency Control.
**Speedy:** This mode keeps latency at 2-3 seconds or lower in most application scenarios and is suitable for live showrooms.
**Smooth:** This mode keeps latency at 5 seconds or lower in most application scenarios and is suitable for application scenarios that require smooth playback but are not sensitive to latency, such as game streaming.



### Step 3. Watermark videos on the client side

Tencent Cloud allows you to watermark videos in the cloud, but this will increase latency by 1-2 seconds. Therefore, if you use the MLVB SDK, we recommend you watermark videos at the host end instead of in the cloud to reduce latency.

### Step 4. Check third-party publishers

We guarantee superior streaming experience via our integrated solution, but if you use third-party software to publish streams, we recommend that you compare your publisher with Tencent Cloud's using the trial demo of the MLVB SDK to see if your publisher is the cause of high latency. Many third-party publishers tend to keep increasing the buffer size to mitigate the problem of low upstream bandwidth.

### Step 5. Check OBS settings

If you use OBS to publish streams and experience high latency, check your configuration against Push via OBS. Make sure you set the keyframe interval to 1 or 2 seconds.

### Step 6. Use LEB

If none of the above solves your problem, you can try using Tencent Cloud's LEB service, which features lower latency than LVB and offers streaming with millisecond latency. For details, please see Live Event Broadcasting (LEB).

# Publish Failure

Last updated：2024-01-13 15:49:41

If you follow the steps in Best Practice > CSS Push but fail to publish streams, check the common reasons for publishing failure listed in this document to troubleshoot the issue.

### 1. Check whether you have configured for your domain name a CNAME record that points to a Tencent Cloud address

Publishing can succeed only if your domain name has a CNAME record that points to a Tencent Cloud address. You can check whether a publishing domain name created has a CNAME record in the **CNAME** column in **Domain Management**.
If your domain name does not have a CNAME record, you can add one for it by following the steps in Configuring CNAME for Domain Name.

### 2. Check whether the network is normal

RTMP publishing uses the **1935** port by default. If the port is not open in the firewall of the network you use for testing, you will be unable to connect to the server. You can check whether this is what caused your publishing failure by switching to another network, for example, 4G.

### 3. Check whether the validity period of the publishing URL is too short.

Some clients may be too cautious about `txTime`. For example, they may set `txTime` to 5 minutes from the current time to prevent traffic theft. This is unnecessary given the presence of the `txSercet` signature. If the validity period is too short, if a host is disconnected during a live stream, he or she may be unable to resume publishing due to the expiration of the publishing URL.
You are advised to set `txTime` to 12 or 24 hours from the current time, longer than an average live streaming session.

### 4. Check if `txSecret` is correct

To ensure security, Tencent Cloud requires configuring hotlink protection for all push URLs and **rejects** all hotlink protection URLs that have expired or are miscalculated. If a push is rejected, the LVB SDK will throw a **PUSH_WARNING_SERVER_DISCONNECT** event.
See Best Practice > CSS Push for how to get reliable publishing URLs.

### 5. Check whether the publishing URL is in use

A publishing URL can be used by only one client at a time. A second client trying to publish using the URL will be rejected by Tencent Cloud. You can log in to the CSS console and check whether a stream is already being published in **Stream Management** > **Live Streams**. You can also check whether a stream is disabled in **Disabled Streams**.

### 6. Why does a `-2` error occurs when I call `startPush` in `V2TXLivePusher` to publish streams?

A `-2` error may occur in the following cases:

Using `V2TXLivePusher` to publish `trtc://` streams in LiteAVSDK_Smart. Only LiteAV_All and LiteAV_Enterprise support the TRTC protocol. LiteAVSDK_Smart does not.

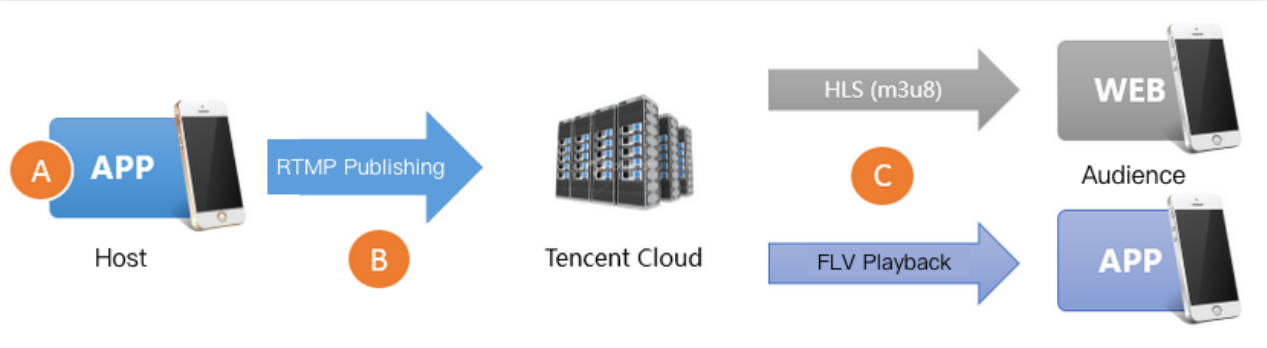
A required parameter is missing from the URL passed in to `startPush`. For how to splice publishing URLs, please see Publishing/Playback URL.

The `V2TXLiveMode_RTC` mode is selected during initialization of the player, but an `rtmp://` URL is passed in.

# Playback Failure

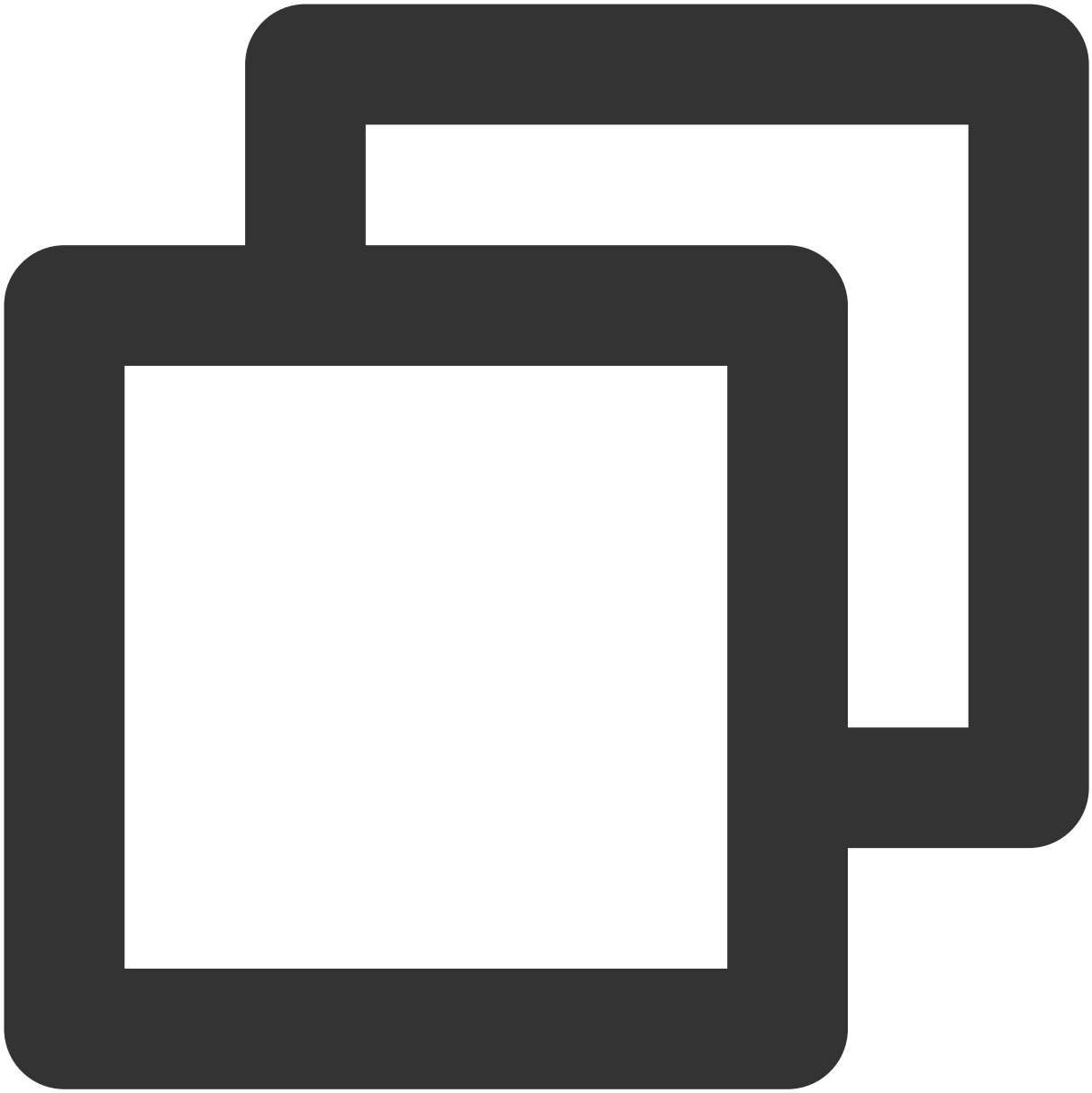
Last updated：2024-01-13 15:49:41

If you are unable to watch a live stream and do not know where the problem lies, follow the steps below to locate the problem. This usually takes less than a minute.




### 1. Check your playback URL

First, check whether your playback URL is correct. Incorrect URLs are the most common cause of playback failure. Tencent Cloud uses publishing and playback URLs for live streaming. Make sure that you are not **using a publishing URL for playback**.



```
rtmp://domain/AppName/StreamName?txSecret=Md5(key+StreamName+hex(time))&txTime=hex(
http://domain/AppName/StreamName.m3u8?txSecret=Md5(key+StreamName+hex(time))&txTime
http://domain/AppName/StreamName.flv?txSecret=Md5(key+StreamName+hex(time))&txTime=
https://domain/AppName/StreamName.m3u8?txSecret=Md5(key+StreamName+hex(time))&txTim
https://domain/AppName/StreamName.flv?txSecret=Md5(key+StreamName+hex(time))&txTime
```

**Note:**

`domain` is the domain name for publish/playback. `AppName` and `StreamName` are custom values, and the default value for `AppName` is `live`. If you do not enable publishing or playback authentication, a URL ends before "?". For example, if the publishing domain name is `www.push.com`, the application name `live`, and the

stream name `test01` , and publishing authentication is not enabled, the publishing URL would be `rtmp://www.push.com/live/test01` .

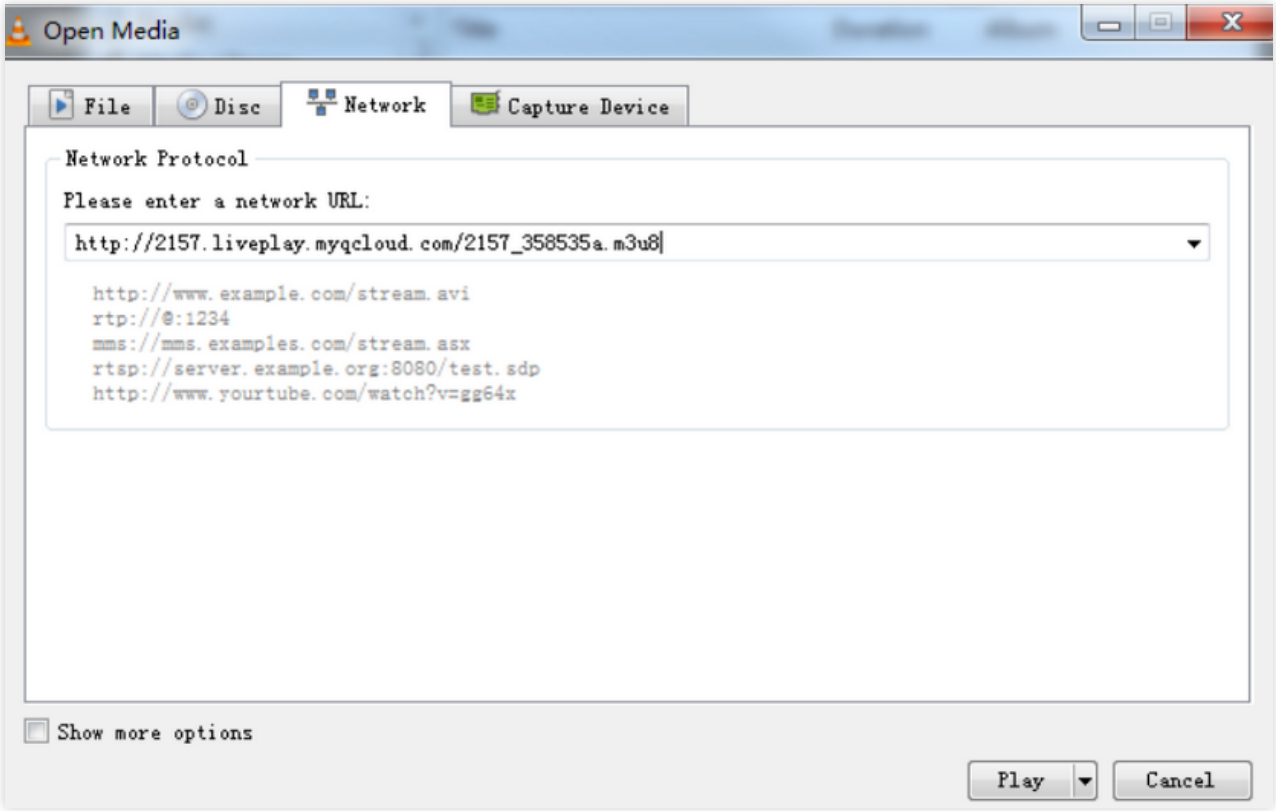
## 2. Check video streams

A correct playback URL does not guarantee successful playback. You need to check the video stream as well.

In **live streaming**, a playback URL becomes unavailable once the host stops publishing streams.

In **VOD**, you cannot watch a video if the video file has been removed from the cloud.

A common solution is to run a check using VLC, which is an open-source player for PC that supports a wide range of protocols. If you use Tencent Cloud's private WebRTC protocol, you can run a check using the trial demo.




## 3. Check the playback end

If there's no problem with the video stream, the next step is checking whether the player is normal.

**Web browser**

**Format**: Mobile browsers support only playback URLs in **HLS (M3U8) or MP4** format.

**HLS (M3U8)**: Tencent Cloud adopts a lazy start mechanism for HLS transcoding. That is to say, it starts transcoding video to the HLS format only after a viewer requests playback via an HLS URL. This is to prevent the waste of resources, but it also brings a problem: **playback via an HLS URL is possible only 30 seconds after the first user requests the playback worldwide**.

**Tencent Cloud web player:** Tencent Cloud's web player supports playback URLs of different protocols and can select the best playback strategy for a platform (PC/Android/iOS). Its selective retry logic also offers a solution to the lazy start issue of the HLS (M3U8) protocol.

**RTMP SDK**

If playback is possible with the RTMP SDK demo, we recommend that you check your integration logic against the playback documents iOS and Android.

### 4. Check firewall restrictions

Firewall restrictions are a common cause of playback failure. Many companies' office networks set restrictions against video streaming. This is achieved by having the firewall check whether an HTTP request involves streaming media. If you can watch live streams using 4G networks but not using your company's office Wi-Fi, then the problem lies in firewall restrictions. Contact your company's IT department and see if they can lift the restrictions for your IP address.

### 5. Check the publishing end

If your playback URL is not playable and the firewall restrictions described in step 4 are nonexistent, the problem probably lies in publishing failure. To troubleshoot the issue, see Troubleshooting Push Failure.

# Instant Streaming

Last updated：2024-01-13 15:49:41

## What Is "Instant Streaming"?

**Instant streaming** means keeping the time it takes for the first video frame to reach audience after playback starts to a few hundred milliseconds.
This is mainly achieved through the optimization of cloud services and the adaptation of the player. If you use LiteAVSDK together with Tencent Video Cloud to implement live streaming, the video loading time can be cut to around **200 ms** or even lower in case of excellent downstream network conditions.

## How to Achieve Instant Streaming?

### App

You can use the MLVB SDK plus the FLV playback protocol to achieve instant streaming.
**HTTP-FLV playback protocol**
HTTP-FLV is currently the most widely used playback protocol in the streaming industry. Because it organizes data in simple formats, audio and video data can be obtained the moment server connection is set up. In contrast, with RTMP, due to the several handshakes required for connection, its streaming speed is slower than FLV.
**Tencent Cloud LiteAVSDK**
The way instant streaming is achieved on the cloud side is quite simple. With a GOP (containing at least one keyframe) always cached on the server, the player can obtain a keyframe (I frame) right after it connects to the server, and can then decode and play the frame. However, there's a downside to caching GOPs in the cloud. The player may be overwhelmed with audio and video data of a few seconds upon connection, resulting in marked playback latency. In addition to instant streaming, a good player should be capable of **latency control**, that is, automatically adjusting playback latency to an acceptable level (e.g., within 1s) without compromising playback experience. Tencent Cloud's LiteAVSDK performs well in this respect. It even allows you to choose a latency control mode based on your needs. For details, please see iOS and Android.

### Desktop browser

Most desktop browsers use Flash Player for video playback (Chrome supports MSE now, but it does not have a notable advantage against Flash Player). The playback policy of Flash Player involves **forced caching**, making it difficult for it to keep the video loading time within 1s. This is evident from the performance of mainstream video websites and live streaming platforms on desktop browsers.

## Mobile browser

### iPhone

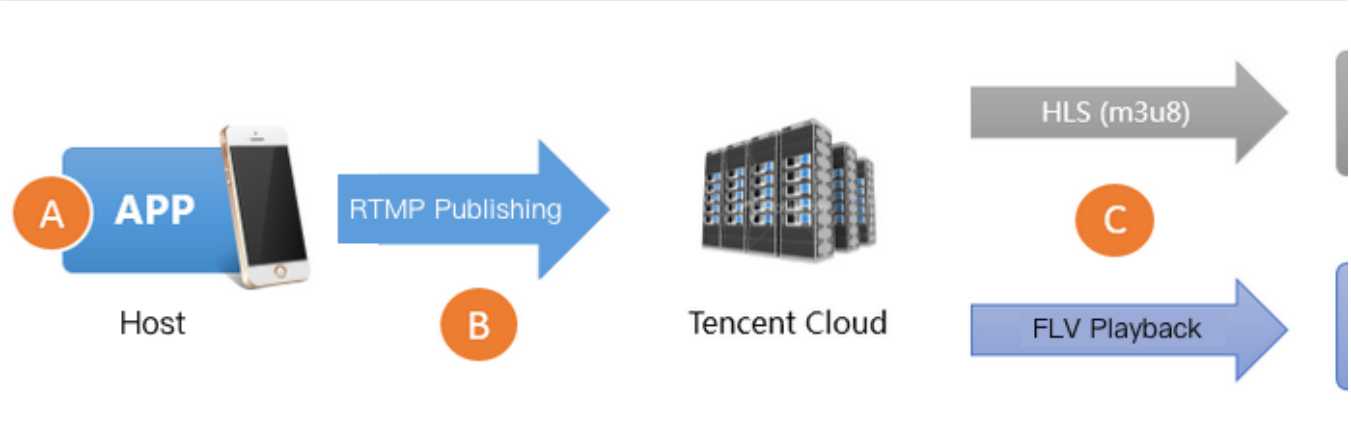
Safari works well with HLS (M3U8). It uses iPhone's decoding chip to facilitate video playback. Usually, you do not have to worry about the video loading speed as long as DNS caches are available, but this is limited to the iOS platform.

### Android

There is great uncertainty to streaming performance on Android, which varies with browser and browser version. For example, QQ Browser and WeChat’s built-in browser use Tencent’s proprietary X5 engine and therefore differ significantly from other browsers in terms of instant streaming.

# Video Stutter (V1)

Last updated：2024-01-13 15:49:41




There are three main reasons for playback stuttering.

**Reason 1: low upstream frame rate**

Low upstream frame rate may be a result of poor performance of the host's mobile phone or the running of CPU-intensive apps in the background. Normally, to ensure smooth playback, the upstream frame rate must be 15 fps or higher. Frame rate lower than 10 fps is deemed **too low**, which will cause **all audience** to experience stuttering. However, when the host's video image changes little, for example, when the host is showing a static image or PowerPoint, low frame rate will not cause stuttering.

**Reason 2: upstream congestion**

The host's mobile phone keeps publishing audio and video data during live streaming. If the phone's upstream bandwidth is too low, the data waiting to be published will accumulate, causing congestion in data transfer and stuttering playback experience for **all audience**.

Even though **ISPs in the Chinese mainland** offer broadband packages with downstream bandwidth as high as 10 Mbps or 20 Mbps, or even 100 Mbps or 200 Mbps, upstream bandwidth remains small. In many small cities, upstream bandwidth is up to 512 Kbps, which means that a maximum of 64 KB data can be uploaded per second.

**Wi-Fi** uses the carrier-sense multiple access and collision avoidance (CSMA/CA) strategy specified in IEEE 802.11. To put it simply, a Wi-Fi hotspot can communicate with only one phone at a time, and other phones must query if communication is possible before initiating a connection to a hotspot. Therefore, the more people using a Wi-Fi hotspot, the slower the connection is. Furthermore, Wi-Fi signals are weakened significantly when passing through walls. Among average Chinese households, few take this into consideration when designing and decorating their houses, and hosts probably pay little attention to how many walls they are apart from their routers when they stream at home.

**Reason 3: poor downstream connection**

Poor downstream connection means slow download speed or instable network for audience. A bitrate of 2 Mbps in live streaming means 2 Mb of data needs to be downloaded per second. Audience will experience stuttering if their network bandwidth is low, but users with sufficient bandwidth will not.

# Checking SDK Status Metrics

The MLVB SDK of Tencent Video Cloud Toolkit has a status feedback mechanism that reports status metrics of the SDK every 1-2 seconds. If you use the MLVB SDK to push streams, you can register TXLivePushListener to get the metrics. The metrics reported are as follows:

| Items | Description |
| --- | --- |
| NET_STATUS_CPU_USAGE | CPU usage of the current process and the device |
| NET_STATUS_VIDEO_FPS | Current video frame rate, i.e., the number of frames produced by the video encoder per second |
| NET_STATUS_NET_SPEED | Current data transmission speed (Kbps) |
| NET_STATUS_VIDEO_BITRATE | Output bitrate of the video encoder, i.e., the amount of video data produced by the encoder per second (Kbps) |
| NET_STATUS_AUDIO_BITRATE | Output bitrate of the audio encoder, i.e., the amount of audio data produced by the encoder per second (Kbps) |
| NET_STATUS_CACHE_SIZE | Accumulated audio/video data size. A value ≥ 10 indicates that there isn't enough upstream bandwidth to handle the audio/video data generated. |
| NET_STATUS_CODEC_DROP_CNT | Number of packet drops globally. To prevent data accumulation from becoming increasingly worse, the SDK starts dropping packets once the accumulated data exceeds a certain threshold. The higher number of packets the SDK drops, the severer the network problem is. |
| NET_STATUS_SERVER_IP | IP address of the server connected for push, which is typically the one with the fewest hops from the client. |

# Fixing Low Frame Rate

### 1. How to know when the frame rate is too low

You can learn about the video frame rate of the current push from the **VIDEO_FPS** status parameter returned by the MLVB SDK through TXLivePushListener. Normally, the frame rate must be 15 FPS or higher to ensure smooth playback. Viewers usually experience notable lag when the push frame rate is lower than 10 FPS.

### 2. How to fix the problem

#### 2.1 Tracking `CPU_USAGE`

You can learn about the **CPU usage of the stream pushing SDK and the entire system** from the **CPU_USAGE** status parameter returned by the MLVB SDK through TXLivePushListener. If the system CPU usage exceeds 80%, the capturing and encoding of video data will be affected; if the CPU usage reaches 100%, it is difficult to even ensure smooth pushing by hosts, let alone superior watching experience for viewers.

#### 2.2 Identifying the biggest CPU consumers

The stream pushing SDK is not the only CPU consumer in a live streaming app. Leaving on-screen comments, sending hearts, and text messaging all consume CPU. To monitor and evaluate the CPU usage of the stream pushing SDK only, you may use our Basic Edition Demo.

#### 2.3 Choosing a reasonable resolution

High resolution does not necessarily result in high video quality. To begin with, high resolution translates into improved video quality only when the bitrate is high too. Low bitrate and high resolution usually produce lower video quality than high bitrate and low resolution. In addition, viewers may be able to sense notable differences between a resolution of 1280 x 720 pixels and 960 x 540 pixels when watching videos full screen on PCs, but not on mobile phones, whose average screen size is only around 5 inches. High resolution increases the CPU usage of the SDK significantly. Therefore, you are advised to set setVideoQuality in TXLivePusher of the MLVB SDK to **High Definition**. You may not get the high video quality expected by setting the resolution too high.

#### 2.4 Using hardware for acceleration if necessary

Most smartphones today use hardware encoders to reduce the CPU consumption of video encoding. When the CPU usage of your app is too high, you can enable hardware encoding to lower the usage. When `setVideoQuality` of TXLivePusher is set to **High Definition**, software encoders are used by default (hardware encoding doesn't work well on some Android devices and results in blurry video). You can use `enableHWAcceleration` of TXLivePushConfig to enable hardware encoding.

## Fixing Upstream Congestion

Statistics show that upstream congestion at the host end is responsible for over 80% of playback lag in cloud live streaming.

### 1. How to know when there is upstream congestion

**1.1: Relationship between `BITRATE` and `NET_SPEED`** `BITRATE` ( `VIDEO_BITRATE` and `AUDIO_BITRATE` ) is the amount of audio/video data produced by the encoder per second for push, and

`NET_SPEED` is the amount of data actually pushed per second. If `BITRATE` == `NET_SPEED` most of the time, the push quality is excellent. However, if `BITRATE` >= `NET_SPEED` for a long period of time, the push quality is unsatisfactory.

**1.2: `CACHE_SIZE` and `DROP_CNT`**

When `BITRATE` >= `NET_SPEED` , the audio/video data produced by the encoder builds up on the host's phone. `CACHE_SIZE` indicates the severity of the accumulation of data. Once it exceeds the warning threshold, the SDK will start dropping data, which increases `DROP_CNT` . The figures below are a typical example of upstream congestion. As you can see, `CACHE_SIZE` stays above the **warning threshold** (red line) throughout the course, which indicates that the upstream network fails to meet the demand for data transmission, leading to serious upstream congestion.

**Note:**

You can find the above figures in **LVB console** > **Statistics** > **Operation Analysis**.

## 2. How to fix the problem

### 2.1 Informing hosts of poor network conditions

In scenarios where video quality is highly valued, it is advisable to inform the hosts of bad network conditions through UI notifications. For example, you may send this notification to hosts: **Bad network conditions. Please move closer to your router or make sure that your Wi-Fi signal does not have to pass through walls.**

For more information on how to do this, please see **Documentation > Mobile Live Video Broadcasting > Basic Features > Camera Push > Event Handling**. You are advised to remind hosts to check their network conditions if your app receives the **PUSH_WARNING_NET_BUSY** event multiple times within a short period of time. This is because hosts are often unable to notice the problem of upstream congestion until reminded by the app or viewers.

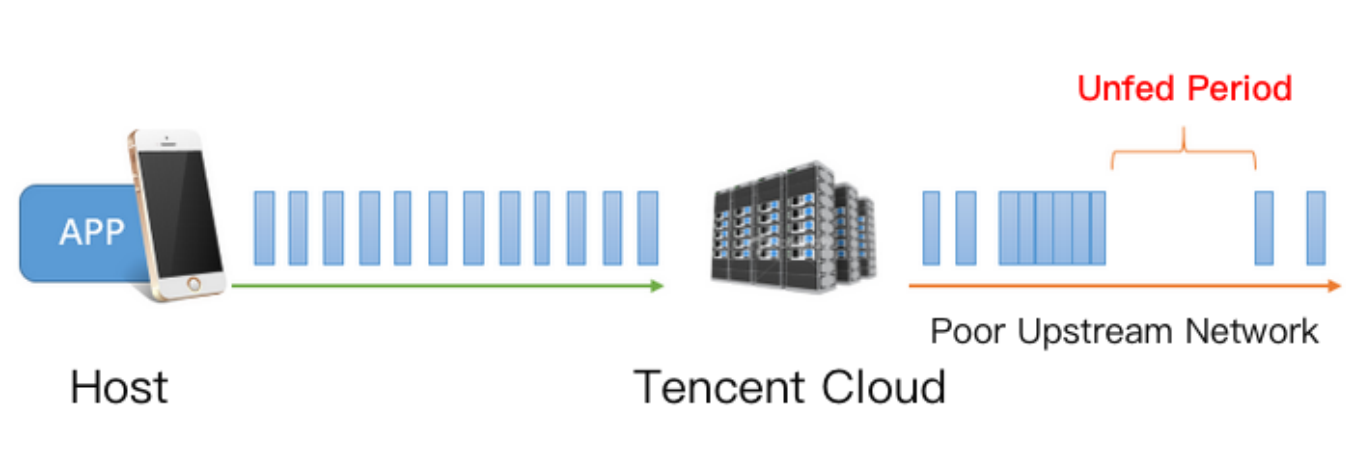
### 2.2 Setting appropriate values for encoding parameters

We recommend the following encoding settings through the `setVideoQuality` API in TXLivePusher.

| Setting | Resolution | FPS | Bitrate | Use Case |
| --- | --- | --- | --- | --- |
| Standard definition | 360 x 640 | 15 | 400-800 Kbps | Use this setting for cost-sensitive scenarios. It may result in videos that lack clarity, but its bandwidth cost is 60% lower than that of high definition. |
| High definition (recommended) | 540 x 960 | 15 | 1200 Kbps | Use this setting for scenarios with high requirements on video quality. It guarantees clear video images on most mainstream mobile phones on the market. |
| Ultra high definition | 720 x 1280 | 15 | 1800 Kbps | Use this setting with caution. It is recommended only if viewers watch live broadcasts on big screens and hosts have |

| | | | | |
|---|---|---|---|---|
| | | | | excellent network connections, but not if viewers watch mostly on small screens. |

# Fixing Player-End Issues




### 1. Lag and latency

As you can see from the figure above, both downstream network fluctuations and insufficient downstream bandwidth can result in **unfed periods** (during which the app cannot get any audio/video data for playback) in the playback process. To avoid playback lag, the app needs to buffer video data enough to cover the unfed periods. However, buffering too much data causes a new problem: **high latency**, which is undesirable for scenarios that stress host-viewer interaction. The latency could **build up** over time if not fixed, meaning that it increases as the playback continues. The ability to fix latency is a key performance indicator for players. **Latency and playback smoothness are like the two ends of a scale**. To ensure low latency, you may have to compromise network stability, which causes notable playback lag, and to ensure smooth playback, you must deal with high latency. A typical example is the introduction of a 20-30 second delay in the playback of HLS (m3u8) URLs to ensure watching experience.

### 2. How to fix the problem

Through optimization across multiple versions of the MLVB SDK, we have developed an automatic adjustment technology, based on which we came up with three latency control schemes, allowing you to deliver superior playback experience even without much knowledge about bandwidth control or processing.

**Auto mode**: use this mode if you are not sure about what scenarios you will deal with.

**Note:**

You can switch to this mode by setting `setAutoAdjustCache` to `true` in TXLivePlayConfig. In this mode, the player adjusts latency automatically based on network conditions to minimize the latency between hosts and viewers

and consequently ensure host-viewer interaction quality while delivering decent playback experience. By default, the player adjusts latency in the range of 1-5 seconds. You can use `setMinCacheTime` and `setMaxCacheTime` to modify the default range.

**Speedy mode**: suitable for **live show streaming** and other scenarios that require low latency.

**Note:**

You can switch to this mode by **setting both `SetMinCacheTime` and `setMaxCacheTime` to one second**. The auto and speedy mode differ only in terms of the `MaxCacheTime` value, which is generally lower in the speedy mode and higher in the auto mode. This flexibility is made possible by the automatic control technology of the SDK, which can automatically adjust latency without causing lag. `MaxCacheTime` is associated with the adjustment speed. The higher the `MaxCacheTime` value, the more conservative the adjustment is, and the less likely lag will occur.

**Smooth mode**: suitable for **live game streaming** and other high-bitrate and high-definition scenarios.

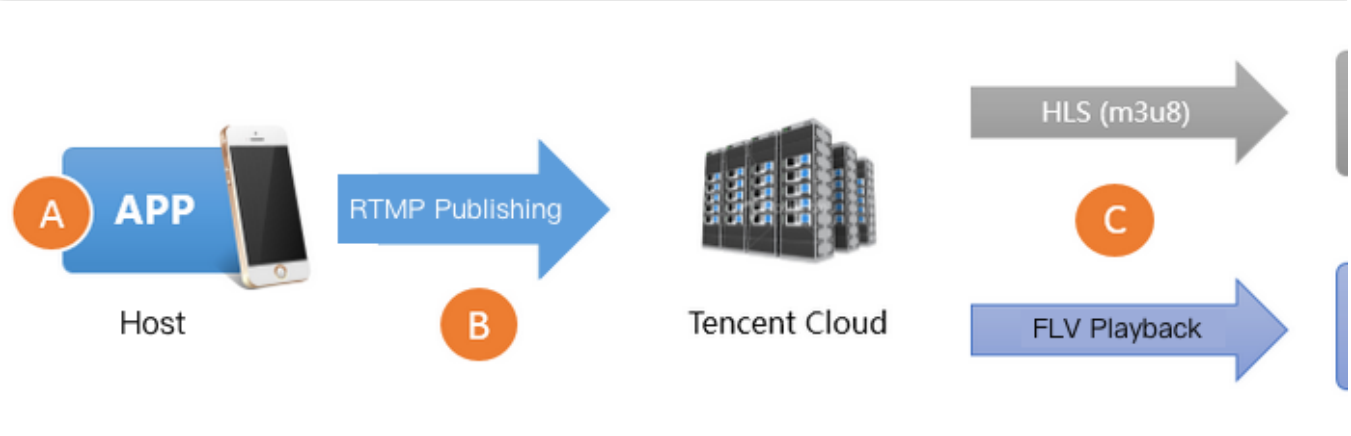
**Note:**

You can switch to this mode by setting `setAutoAdjustCache` of the player to `false` . In this mode, the player adopts a strategy similar to the buffering strategy of Adobe Flash Player for IE. When lag occurs, the player switches to the loading mode, and when enough data is buffered; it returns to the playing mode, until the next network fluctuation. By default, 5 seconds of video data is buffered, which you can change via `setCacheTime` .

This seemingly simple mode, which reduces playback lag by moderately increasing latency, is the more reliable option for scenarios that are not demanding on low latency.

# Video Stutter (V2)

Last updated：2024-01-13 15:49:41




There are three main reasons for playback stuttering.

**Reason 1: low upstream frame rate**

Low upstream frame rate may be a result of poor performance of the host's mobile phone or the running of CPU-intensive apps in the background. Normally, to ensure smooth playback, the upstream frame rate must be 15 fps or higher. Frame rate lower than 10 fps is deemed **too low**, which will cause **all audience** to experience stuttering. However, when the host's video image changes little, for example, when the host is showing a static image or PowerPoint, low frame rate will not cause stuttering.

**Reason 2: upstream congestion**

The host's mobile phone keeps publishing audio and video data during live streaming. If the phone's upstream bandwidth is too low, the data waiting to be published will accumulate, causing congestion in data transfer and stuttering playback experience for **all audience**.

Even though **ISPs in the Chinese mainland** offer broadband packages with downstream bandwidth as high as 10 Mbps or 20 Mbps, or even 100 Mbps or 200 Mbps, upstream bandwidth remains small. In many small cities, upstream bandwidth is up to 512 Kbps, which means that a maximum of 64 KB data can be uploaded per second.

**Wi-Fi** uses the carrier-sense multiple access and collision avoidance (CSMA/CA) strategy specified in IEEE 802.11. To put it simply, a Wi-Fi hotspot can communicate with only one phone at a time, and other phones must query if communication is possible before initiating a connection to a hotspot. Therefore, the more people using a Wi-Fi hotspot, the slower the connection is. Furthermore, Wi-Fi signals are weakened significantly when passing through walls. Among average Chinese households, few take this into consideration when designing and decorating their houses, and hosts probably pay little attention to how many walls they are apart from their routers when they stream at home.

**Reason 3: poor downstream connection**

Poor downstream connection means slow download speed or instable network for audience. A bitrate of 2 Mbps in live streaming means 2 Mb of data needs to be downloaded per second. Audience will experience stuttering if their network bandwidth is low, but users with sufficient bandwidth will not.

# Checking SDK Performance Metrics

The MLVB SDK of Tencent Video Cloud Toolkit has a feedback mechanism that reports different performance metrics every 2 seconds. If you use the MLVB SDK for publishing, you can register a V2TXLivePusherObserver listener and get the statistics in the `onStatisticsUpdate` callback. The table below lists the metrics included in `V2TXLivePusherStatistics` and their meanings.

| Metric | Description |
| --- | --- |
| appCpu | CPU usage of the app (%) |
| systemCpu | CPU usage of the system (%) |
| width | Video width |
| height | Video height |
| fps | Frame rate (fps) |
| audioBitrate | Audio bitrate in Kbps |
| videoBitrate | Video bitrate in Kbps |

# Fixing Low Frame Rate

### 1. How to know whether the frame rate is too low

The **V2TXLivePusherStatistics.fps** field in the onStatisticsUpdate callback of `V2TXLivePusherObserver` indicates the frame rate for push. Normally, the frame rate must be 15 FPS or higher to ensure smooth playback. Viewers usually experience obvious stuttering when the push frame rate is lower than 10 FPS.

### 2. How to fix the problem

#### 2.1 Monitoring `appCpu` and `systemCpu`

You can learn about the **CPU usage of the app and system** from the **V2TXLivePusherStatistics.appCpu** and **V2TXLivePusherStatistics.systemCpu** fields in the onStatisticsUpdate callback of

`V2TXLivePusherObserver`. If the system CPU usage exceeds 80%, both video capturing and encoding may be affected; if it reaches 100%, it is difficult to even ensure smooth publishing, let alone playback experience for audience.

**2.2 Identifying CPU consumers**

The MLVB SDK is not the only CPU consumer in a live streaming application. Leaving on-screen comments, sending hearts, and text messaging all consume CPU. To monitor the CPU usage of the MLVB SDK only, you may use the demo.

**2.3 Choosing an appropriate resolution**

High resolution does not necessarily result in high video quality. To begin with, high resolution translates into improved video quality only when the bitrate is also high. Low bitrate and high resolution usually produce lower video quality than high bitrate and low resolution. In addition, audience may be able to sense obvious differences between a resolution of 1280 x 720 px and 960 x 540 px when watching videos full screen on PCs, but not on mobile phones, whose average screen size is only around 5 inches. High resolution increases the CPU usage of the SDK significantly. Therefore, you are advised to set video quality to **HD** using the setVideoQuality API in `V2TXLivePusher` of the MLVB SDK. You may not get the high video quality expected by setting the resolution too high.

# Fixing Upstream Congestion

Statistics show that upstream congestion at the host end is responsible for over 80% of playback stuttering in live streaming.

### 1. Informing hosts of poor network conditions

In scenarios where video quality is important, you are advised to inform the hosts of bad network conditions through UI notifications. For example, you may send this notification to hosts: **Bad network conditions. Please move closer to your router or make sure that your Wi-Fi signal does not have to pass through walls.**

You can refer to the **Event Handling** section in the document about the publishing feature of the MLVB SDK to achieve this. If your app receives the V2TXLIVE_WARNING_NETWORK_BUSY event callback multiple times in a short period of time, remind hosts to check their network conditions. Hosts are often unable to notice upstream congestion until reminded by audience or an app notification.

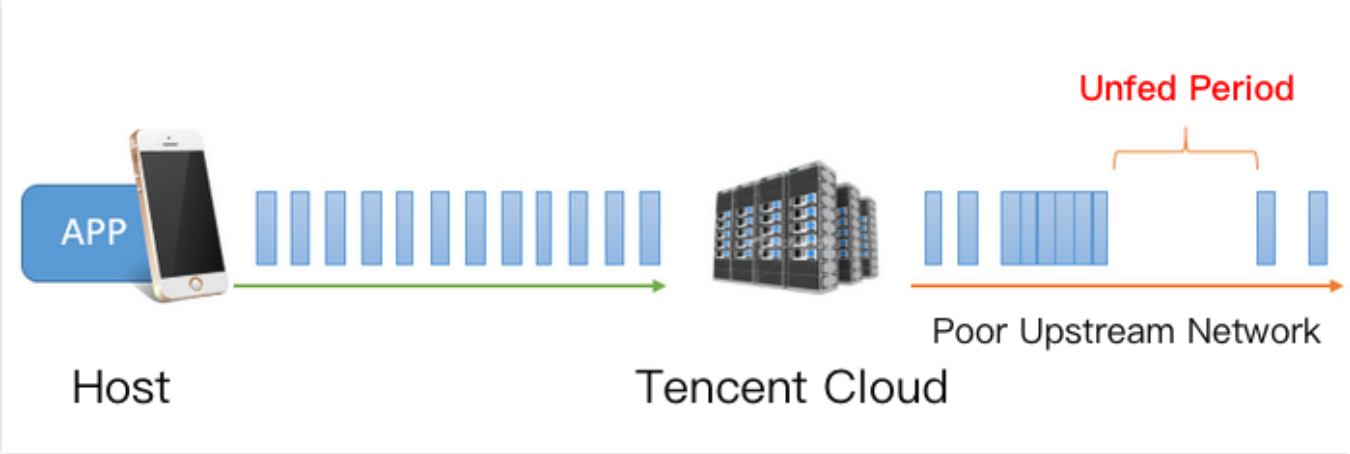
### 2. Using the recommended encoding settings

You can set encoding parameters using the setVideoQuality API in `V2TXLivePusher`. We recommend the following encoding settings.

| Scenario | `resolution` | `resolutionMode` |
| --- | --- | --- |
| Live showroom | V2TXLiveVideoResolution960x540<br>V2TXLiveVideoResolution1280x720 | Landscape or portrait |
| Live game streaming | V2TXLiveVideoResolution1280x720 | Landscape or portrait |

| | | |
|---|---|---|
| Mic connect (primary image) | V2TXLiveVideoResolution640x360 | Landscape or portrait |
| Mic connect (small image) | V2TXLiveVideoResolution480x360 | Landscape or portrait |
| Blu-ray live streaming | V2TXLiveVideoResolution1920x1080 | Landscape or portrait |

# Fixing Player-End Issues




### 1. Stuttering and latency

As shown in the figure above, both downstream network fluctuations and insufficient downstream bandwidth can result in **unfed periods** (during which the app cannot get any audio/video data for playback) in the playback process. To avoid playback stuttering, the application needs to cache video data enough to cover the unfed periods. However, caching too much data causes a new problem: **high latency**, which is undesirable for interactive scenarios. The latency could **build up** over time if not fixed, meaning that it increases as the playback continues. The capability to fix latency is a key performance indicator for players. **Latency and playback smoothness are like the two ends of a scale**. To ensure low latency, you may have to compromise network stability, which causes playback stuttering, and to ensure smooth playback, you must deal with high latency. A typical example of the latter is the introduction of a 20-30 second delay in the playback of HLS (M3U8) streams to ensure smooth playback experience.

### 2. How to fix the problem

To allow you to deliver superior playback experience without having to learn QoS control, we have developed an automatic latency control technology, which we optimized from version to version. You can use the setCacheParams API in `V2TXLivePlayer` to select one of three latency control modes.

-**Auto mode**: Use this mode if you are not sure what scenarios you will deal with.

**Note:**

In this mode, the player adjusts latency automatically based on network conditions to minimize the latency between hosts and audience and ensure host-audience interaction quality while delivering smooth playback experience. By default, the player adjusts latency in the range of 1-5 seconds, which you can modify using the `setCacheParams` API.

**Speedy mode**: This mode is suitable for **live showroom** and other interactive scenarios that require low latency.

**Note:**

You can switch to this mode by **setting both `minTime` and `maxTime` to 1 second**. The auto and speedy modes differ only in terms of `maxTime`, whose value is generally lower in the speedy mode. This flexibility is made possible by the SDK's automatic latency control technology, which can automatically adjust latency without causing stuttering. `maxTime` is associated with adjustment speed. The higher the `maxTime` value, the more conservative the adjustment is, and the less likely stuttering will occur.

**Smooth mode**: This mode is suitable for **live game streaming** and other high-bitrate and high-definition scenarios.

**Note:**

In this mode, the player adopts a strategy similar to the caching policy of Adobe Flash Player. When stuttering occurs, the player switches to the loading mode, and when enough data is cached, it returns to the playing mode, until the network fluctuates again. By default, 5 seconds of video data is cached, which you can change using the `setCacheParams` API.

This seemingly simple mode reduces playback stuttering by moderately increasing latency and is therefore the more reliable option for scenarios that do not require low latency.

# Generating UserSig

Last updated：2024-01-13 15:49:41

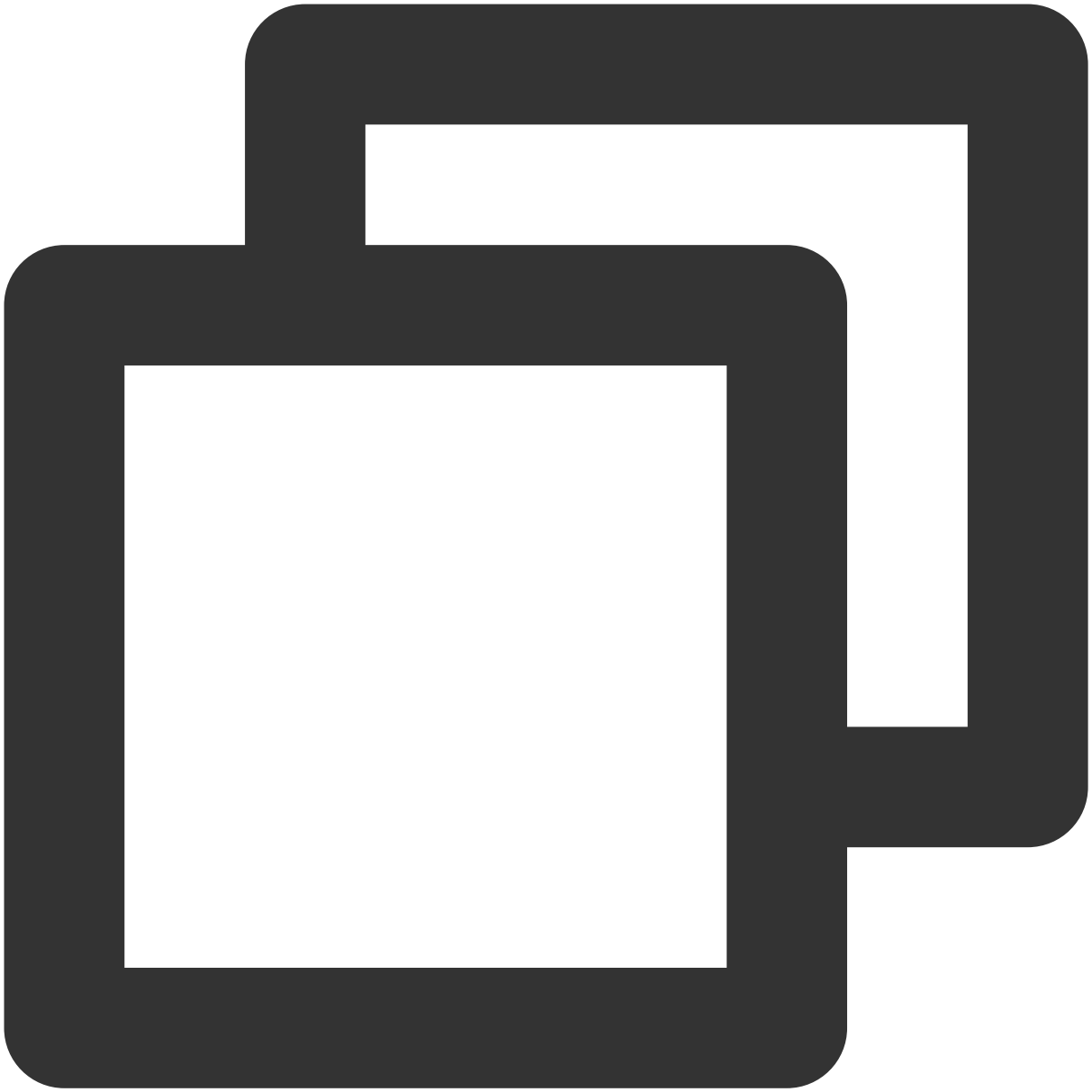
### What is UserSig?

`UserSig` is a security signature designed by Tencent Cloud to prevent attackers from accessing your Tencent Cloud account.

Currently, Tencent Cloud services including MLVB, TRTC, and IM all use this security mechanism. Whenever you want to use these services, you must provide three key pieces of information, i.e., `SDKAppID` , `UserID` , and `UserSig` in the initialization or login function of the corresponding SDK.

`SDKAppID` is used to identify your application, and `UserID` your user. `UserSig` is a security signature calculated based on the two parameters using the **HMAC SHA256** encryption algorithm. Attackers cannot use your Tencent Cloud traffic without authorization as long as they cannot forge a `UserSig` .

See below for how `UserSig` is calculated. Basically, it involves hashing crucial information including `SDKAppID` , `UserID` , and `ExpireTime` .



```
// UserSig formula, in which `secretkey` is the key used to calculate UserSig

usersig = hmacsha256(secretkey, (userid + sdkappid + currtime + expire +
                                 base64(userid + sdkappid + currtime + expire)))
```

**Note:**

`currtime` is the current system time and `expire` the expiration time of the signature.

For more information, see How do I calculate UserSig on the client? and How do I calculate UserSig on the server?.

### How do I obtain a key?

Log in to the CSS console and go to Application Management to view the key required to calculate `UserSig`.

1. Click your application to enter the details page. If there isn't an application yet, create one.
2. Select the **Application Management** tab and click **View Key**.

### How do I calculate UserSig on the client?

We provide an open-source module called `GenerateTestUserSig` in the MLVB SDK sample code. Set the three member variables of `SDKAPPID`, `EXPIRETIME`, and `SECRETKEY`, and call `genTestUserSig()` to generate a `UserSig` and get started quickly with the SDK.

| Language | Platform | Source Code |
| --- | --- | --- |
| Objective-C | iOS | Github |
| Java | Android | GitHub |

**Note:**

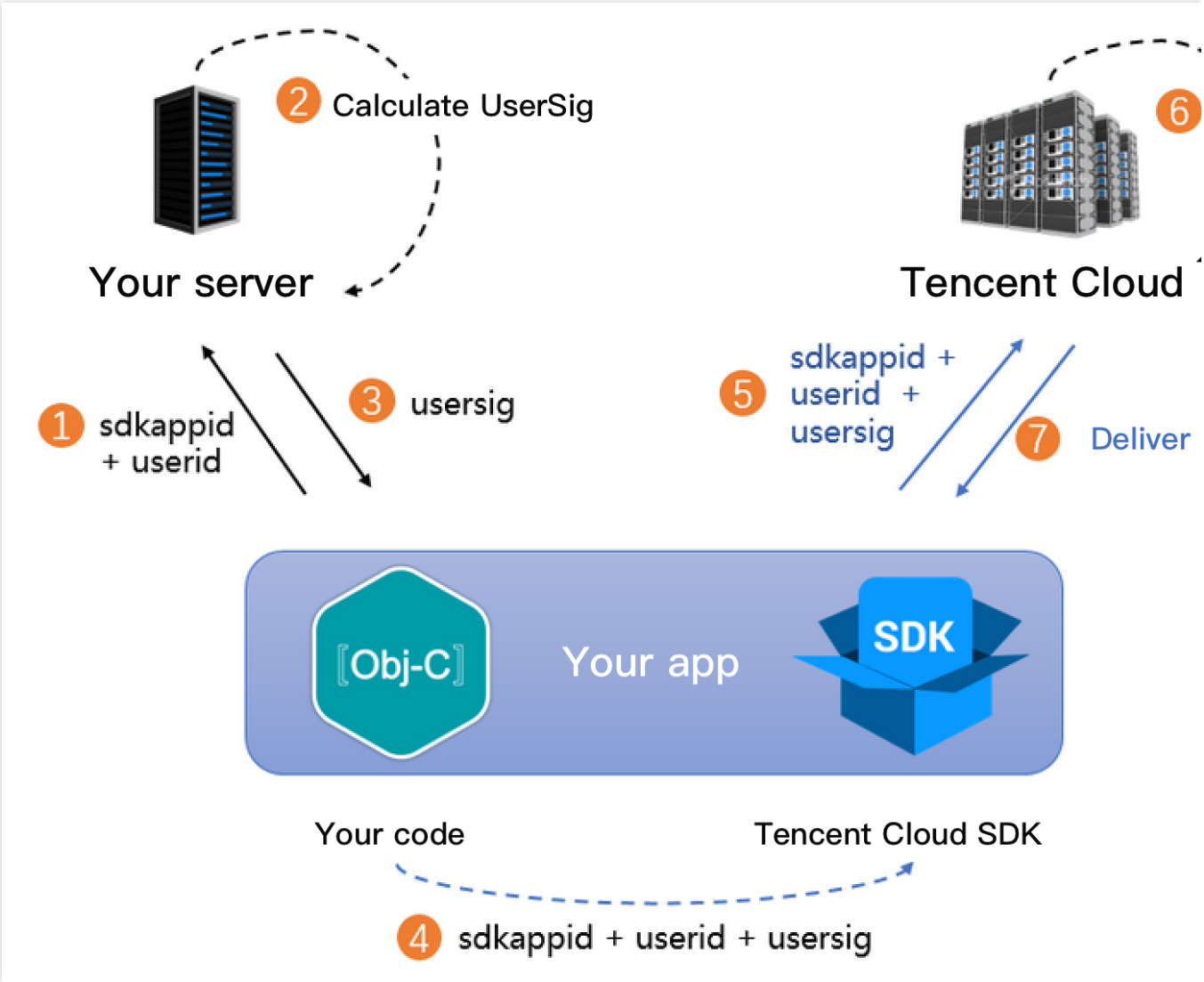
This method is only applicable for debugging. It's **not recommended** for official launch because `SECRETKEY` of the client code (especially on the web) may be easily decompiled and reversed. If your key is leaked, attackers can steal your Tencent Cloud traffic.

The correct method is to deploy the `UserSig` calculation code on your project server so that your application can request from your server a `UserSig` that is calculated whenever one is needed.

### How do I calculate UserSig on the server?

Using the server to calculate `UserSig` offers the utmost protection against key leakage, for it is more difficult to hack a server than it is to reverse engineer an application. See below for the specific method.

1. Before your application calls the SDK initialization function, request `UserSig` from your server.
2. Your server will calculate a `UserSig` based on the `SDKAppID` and `UserID`. The calculation source code is provided above.
3. The sever returns the `UserSig` to your application.
4. Your application sends the `UserSig` to the SDK through a specific API.
5. The SDK submits the `SDKAppID + UserID + UserSig` to the Tencent Cloud server for verification.
6. Tencent Cloud verifies the validity of the `UserSig`.
7. If the `UserSig` is valid, real time audio/video services will be provided to the TRTC SDK.




To simplify your implementation process, we provide `UserSig` calculation source code in multiple languages.

| Programming Language | Signature Algorithm | Key Function | Download Link |
|---|---|---|---|
| Java | HMAC-SHA256 | genUserSig | GitHub |
| GO | HMAC-SHA256 | genUserSig | GitHub |
| PHP | HMAC-SHA256 | genUserSig | GitHub |
| Nodejs | HMAC-SHA256 | genUserSig | GitHub |
| Python | HMAC-SHA256 | genUserSig | GitHub |
| C# | HMAC-SHA256 | genUserSig | GitHub |

### Legacy algorithm

To simplify signature calculation and facilitate your use of Tencent Cloud services, on August 6, 2019, IM switched from the legacy algorithm ECDSA-SHA256 to the new algorithm HMAC-SHA256. This means that all applications created on and after August 6, 2019 will use the HMAC-SHA256 algorithm.
If your application was created before July 19, 2019, you can continue to use the old signature algorithm, whose source code can be downloaded at the links below.

| Programming Language | Signature Algorithm | Download Link |
| --- | --- | --- |
| Java | ECDSA-SHA256 | GitHub |
| C++ | ECDSA-SHA256 | GitHub |
| GO | ECDSA-SHA256 | GitHub |
| PHP | ECDSA-SHA256 | GitHub |
| Nodejs | ECDSA-SHA256 | GitHub |
| C# | ECDSA-SHA256 | GitHub |
| Python | ECDSA-SHA256 | GitHub |

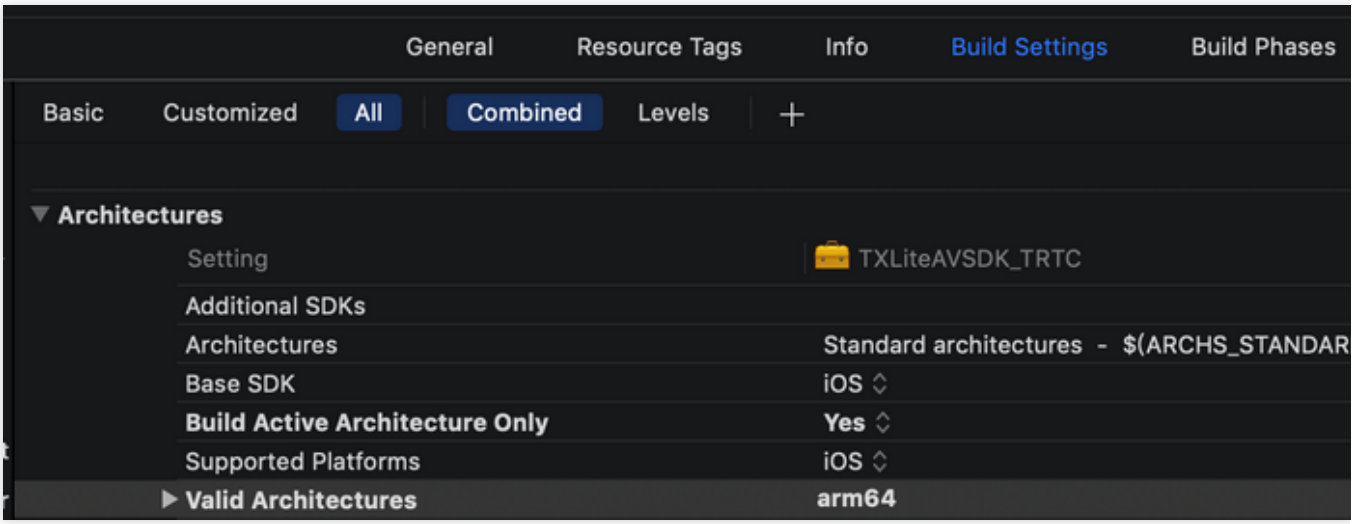
# Downsizing Installation Package

Last updated：2024-01-13 15:49:41

## How do I reduce the size of an installation package for iOS?

### Packaging arm64 only (recommended)

You can package only arm64 into applications for iPhone 5S and later. In Xcode, set **Build Active Architecture Only** to **Yes** and enter `arm64` only for **Valid Architectures**. A single-architecture TRTC SDK adds only 1.9 MB to an IPA file.




## How do I reduce the size of an installation package for Android?

### 1. Packaging only 1 or 2 SO files

If your application is intended for the Chinese mainland, you can package only the SO file for `armeabi-v7a`, in which case the SDK will add 5 MB or less to your installation package. If you want to publish your application on Google Play, then you can package the SO files for `armeabi-v7a` and `arm64-v8a`.
Specifically, you need to add `abiFilters "armeabi-v7a"` in `build.gradle` of your project to package only the `armeabi-v7a` SO file, and `abiFilters "armeabi-v7a", "arm64-v8a"` to package the `armeabi-v7a` and `arm64-v8a` SO files.
If you do not intend to publish your application on Google Play:

```
android {
    compileSdkVersion rootProject.ext.compileSdkVersion
    buildToolsVersion rootProject.ext.buildToolsVersion
    defaultConfig {
        applicationId "com.tencent.liteav.demo"
        minSdkVersion rootProject.ext.minSdkVersion
        targetSdkVersion rootProject.ext.targetSdkVersion
        versionCode 1
        versionName "2.0"
        multiDexEnabled true

        ndk {
            abiFilters "armeabi-v7a"
        }
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles.add(file('proguard-rules.pro'))
        }
    }
```

If you want to publish your application on Google Play:

```
android {
    compileSdkVersion rootProject.ext.compileSdkVersion
    buildToolsVersion rootProject.ext.buildToolsVersion
    defaultConfig {
        applicationId "com.tencent.liteav.demo"
        minSdkVersion rootProject.ext.minSdkVersion
        targetSdkVersion rootProject.ext.targetSdkVersion
        versionCode 1
        versionName "2.0"
        multiDexEnabled true

        ndk {
            abiFilters "armeabi-v7a","arm64-v8a"
        }
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles.add(file('proguard-rules.pro'))
        }
    }
```

**2. Downloading SO files after installation (packaging JAR only)**

SO files take up most of the size of the MLVB SDK for Android. Therefore, if you want the SDK to add 1 MB or less to your installation package, you can have SO files downloaded to users' phones after installation.

**Step 1. Download the SO files**

Download `LiteAVSDK_x.x.xxx.zip` at GitHub, decompress the file, and find the SO files for the architecture

you use. Find the SO files for the architectures you want to use.

**Step 2. Upload the SO files to your server**

Upload the SO files downloaded in Step 1 to your server (or to Tencent Cloud COS) and note the download URL, such as `http://xxx.com/so_files.zip` .

**Step 3. Download the SO files at the SDK's first launch**

Before users use the SDK's features, for example, to play video, show a loading animation on the UI and tell users that modules are being loaded.

While users wait, your application can download the SO files from `http://xxx.com/so_files.zip` and save them in the `files` folder of your application's root directory. Given the possibility of DNS hijacking and file tampering by the carrier, please check the integrity of the SO files after download.

**Step 4. Use an API to load the SO files**

After the SO files are in place, call the `setLibraryPath()` API in the `TXLiveBase` class (the earliest underlying module of LiteAVSDK), setting the SDK's library paths to the target paths of the downloaded SO files. The SDK will load the SO files from the paths and enable related features.

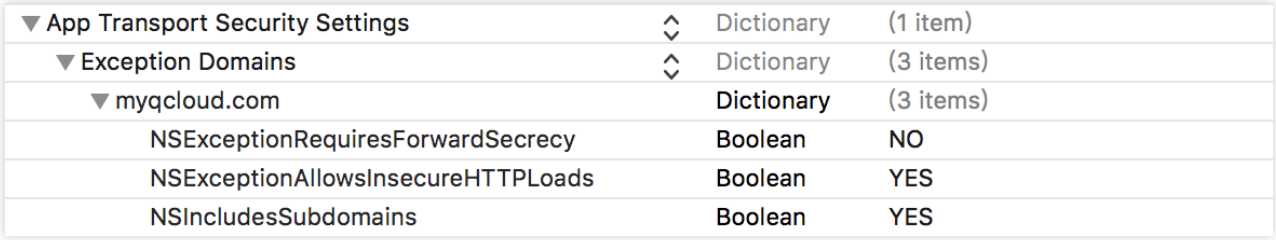
**Note:**

Do not use this method if you want to publish your application on Google Play.

# Apple ATS Requirements

Last updated：2024-01-13 15:49:41

Apple made it clear at WWDC 2016 that all app submitted since January 1, 2017 could no longer use `NSAllowsArbitraryLoads=YES` to bypass ATS restrictions. Tencent Cloud has supported HTTPS. To meet the ATS requirements, you only need to update to the new version of the SDK (no change to APIs was made) and replace the prefix `http://` with `https://` .
It should be noted that HTTPS may provide greater security (which is not that important for video services) than HTTP, but it leads to slower connection speed and higher CPU usage. To continue using HTTP in your app while meeting Apple's new requirements, you can add `myqcloud.com` to `NSExceptionDomains` in `Info.plist` , as shown below.




Apple allows disabling ATS for specific domain names, but you may need to explain to Apple's review team that `myqcloud.com` is used for video playback.