组播 SDK(C 版)说明(V1.0.0)

www.mediapro.cc

一、基本概念

单播、广播、组播:

作为一种与单播（Unicast）和广播（Broadcast）并列的通信方式，组播（Multicast）技术能够有效地解决单点发送、多点接收的问题,从而实现了网络中点到多点的高效数据传送，能够节约大量网络带宽、降低网络负载。利用组播技术可以方便地提供一些新的增值业务，包括在线直播、网络电视、远程教育、远程医疗、网络电台、实时视频会议等对带宽和数据交互的实时性要求较高的信息服务。

三种信息传输方式的比较

1、 单播

如图 1-1 所示，在 IP 网络中若采用单播的方式，信息源（即 Source）要为每个需要信息的主机（即 Receiver）都发送一份独立的信息拷贝。

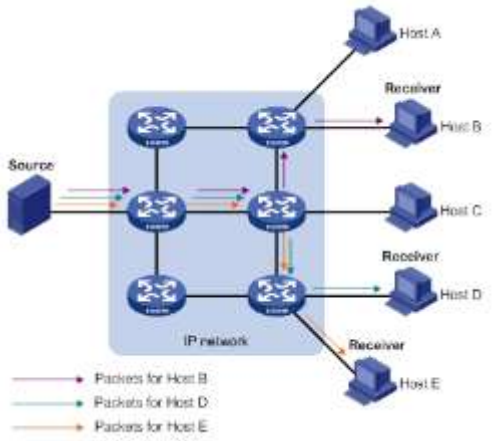

图 1-1 单播

采用单播方式时，网络中传输的信息量与需要该信息的用户量成正比，因此当需要该信息的用户数量较大时，信息源需要将多份内容相同的信息发送给不同的用户，这对信息源以及网络带宽都将造成巨大的压力。从单播方式的信息传播过程可以看出，该传输方式不利于信息的批量发送。单播主要用于公网场景或无线局域网中，这类场景中广播与组播均无法应用或效果不佳。

2、 广播

如图 1-2 所示，在一个网段中若采用广播的方式，信息源（即 Source）将把信息传送给

该网段中的所有主机，而不管其是否需要该信息。

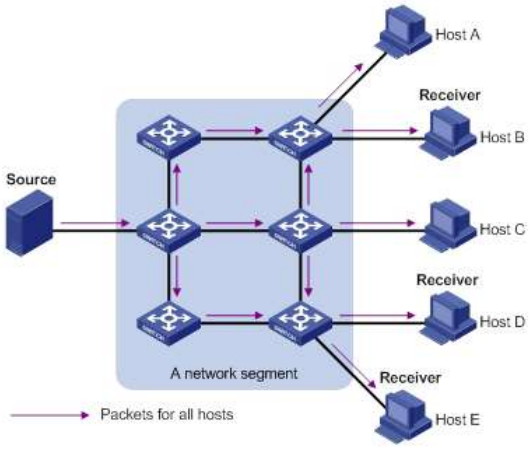

图 1-2 广播

假设只有 Host B、Host D 和 Host E 需要信息，若将该信息在网段中进行广播，则原本不需要信息的 Host A 和 Host C 也将收到该信息，这样不仅信息的安全性得不到保障，而且会造成同一网段中信息的泛滥。因此，广播方式不利于与特定对象进行数据交互，并且还浪费了大量的带宽。

3、 组播

综上所述，传统的单播和广播的通信方式均不能以最小的网络开销实现单点发送、多点接收的问题，IP 组播技术的出现及时解决了这个问题。

如图 1-3 所示，当 IP 网络中的某些主机（即 Receiver）需要信息时，若采用组播的方式，组播源（即 Source）仅需发送一份信息，借助组播路由协议建立组播分发树，被传递的信息在距离组播源尽可能远的网络节点才开始复制和分发。

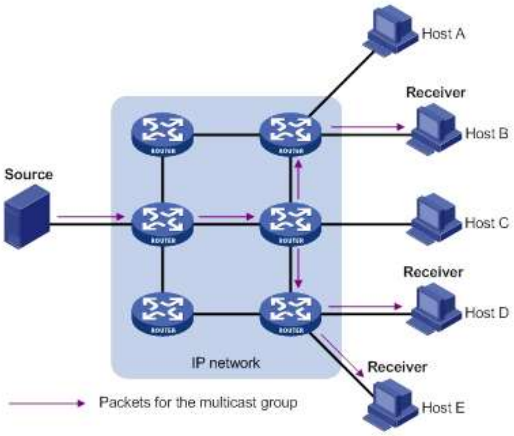

图 1-2 组播

假设只有 Host B、Host D 和 Host E 需要信息，采用组播方式时，可以让这些主机加入同一个组播组（Multicast group），组播源向该组播组只需发送一份信息，并由网络中各路由器根据该组播组中各成员的分布情况对该信息进行复制和转发,最后该信息会准确地发送给 Host B、Host D 和 Host E。

综上所述，组播的优势归纳如下:

相比单播来说，组播的优势在于: 由于被传递的信息在距信息源尽可能远的网络节点才开始被复制和分发,所以用户的增加不会导致信息源负载的加重以及网络资源消耗的显著增加。相比广播来说，组播的优势在于: 由于被传递的信息只会发送给需要该信息的接收者，所以不会造成网络资源的浪费，并能提高信息传输的安全性；另外，广播只能在同一网段中进行，而组播可以实现跨网段的传输。

组播也有自己的劣势:

由于无线组播报文的发送没有相应的确认机制（数据链路层 ACK），因此设备无法对丢失的报文进行重传，导致链路质量差的情况下，无线环境下组播报文丢失严重。

组播传输 SDK: 常见基于组播的音视频应用有同屏教学、校园广播等，组播基于 UDP 传输，因此同样面临 UDP 传输常见的乱序、丢包、重复等挑战，本 SDK 主要为传统 UDP 组播增加 FEC 前向纠错、NACK 丢包重传、Qos、JitterBuff 等功能，为开发音视频组播应用提供快速解决方案。SDK 特点如下:

1、 支持基于 RS 编码的 FEC 前向纠错，专为音视频类应用量身定制。

2、 支持 FEC 动态 Group 分组，避免分组跨帧同时尽可能提升抗连续丢包能力。

3、 发送端传入单帧码流，内部自行拆分拼接，实现透传。

4、 发送端支持 Smooth 平滑发送，对视频大 I 帧场景进行优化。

5、 支持接收端自适应 JitterBuff 缓存技术。

6、 独有的 NACK 组播重传技术，为每个组播接收端提供个性化重传服务。

7、 两套独立 SDK 分别适应不同场景，一套仅负责传输层，外层应用负责编解码。一套负责编解码以及网络传输，外层负责采集、渲染。

8、 C++开发，性能强劲，无外部库依赖，支持 Windows\Android\Linux，支持 32 位、64 位系统。

9、 API 接口简洁易用、完善的文档与 DEMO 参考，持续的技术支持服务。

音视频组播发送端：作为组播源将自身音视频组播发送。

音视频组播接收端：接收音视频组播流。

FEC：前向纠错技术，通过增加冗余带宽方式，提高网络对于丢包的抵抗力。

FEC 相关参数：FEC 相关参数包括 FEC 冗余度方法、FEC 上行冗余度、FEC min group 组大小、FEC max group 组大小。FEC 冗余度方法包括：固定冗余度（本 SDK 仅支持固定冗余度）、自动冗余度两种。固定冗余度即全程使用用户指定的冗余度进行 FEC 编码，自动冗余度则以用户指定的 FEC 冗余度为基础，根据网络情况进行调整，对于网络较好的场合使用低冗余尽量降低带宽。FEC 是分组进行的，即多个拆分包组成一个 group，产生其冗余包。group 越大，同样冗余度的情况下，产生的冗余包越多，抵抗连续丢包能力越强，同时因丢包产生的抖动也会越大（因为 FEC 的 group 可能产生跨多帧的情况，前面包的丢失不得不等到后续包的到来才能恢复）；FEC 分组越大消耗 CPU 资源也越大。建议配置 FEC min group 大小 16；建议根据芯片处理性能设置 FEC max group，在性能足够的设备上建议设置为 64（PC、主流 Android 手机均可设置为 64，嵌入式平台则根据实际情况设置）。

NACK：重传请求机制，与 FEC 配合，在预估 FEC 无法恢复时触发接收端到发送端的重传请求，发送端予以重发响应。NACK 仅重传一次，不保证数据一定传输成功，优点是可以获得稳定的低延时，缺点是画面仍可能因丢包而卡顿。

组播 NACK 支持：组播是一种单向的传输技术，正常情况下无法使用 NACK 重传机制，因为每一个接收端都可能有不同的丢包情况。我们使用单播作为组播的一种补充，从而实现了为不同组播接收端提供个性化的 NACK 服务。

Smooth 平滑：发送端对一帧较大的码流在时间窗口内平滑发送，避免一次性发出给网络带来的压力。

JitterBuff：接收端为了抵消网络传输、丢包恢复、NACK 重传引入的抖动，引入 JitterBuff 缓

存。缓存时间越大,画面流畅度越高,但延时也同步增大。当需要极低延时时,可设置 JitterBuff 为 0。

传输参数: 本 SDK 中传输相关参数包括: 视频通道的上行 FEC 冗余度、上行 FEC Group 分组大小、NACK 重传支持、接收端 JitterBuff 缓存时间。对于音视频组播接收端，同样可以设置上行 FEC 冗余度、上行 FEC Group 分组大小，只是没有实际意义(因为不会发送数据，当然也不会进行 FEC 编码)。

时间戳模式: 本 SDK 提供了内置时间戳（默认）和外部时间戳两种模式，当用户选择外部时间戳模式时，需在发送端提供音视频时间戳（1KHZ 时基，即毫秒单位），该时间戳将在接收端透传露出。当用户选择内置时间戳模式时，SDK 将在用户调用 SDK 发送接口时自动生成时间戳，以简化用户使用。

内部拆包传输: 外层传入一帧码流后,SDK 内部将进行拆分发送,拆分包大小与 FEC min group 取值相关，原则上 400~1000 字节之间波动，拆分后的包大小加上私有头部、UDP\IP 头部后不会超过 1500 字节。

二、API 接口

所有 API 接口定义均位于 SDTerminalMulticastSdk.h 文件中。

1、系统环境初始化，仅需调用一次

```
void  SDTerminalMulticast_Enviroment_Init (const char * outputPath,  int outputLevel)
```

参数:

@param: outputPath: 日志文件输出的目录，若目录不存在，SDK 将自动创建，支持相对路径或绝对路径。日志对于问题定位非常重要，建议开启。

@param: outputLevel: 日志输出的级别，只有等于或者高于该级别的日志会输出到文件，日志级别取值见 LOG_OUTPUT_LEVEL_MULTICAST，有 DEBUG、INFO、WARNING、ERROR、ALARM、FATAL、NONE 几个级别可选，当指定为 NONE 时，将不会生成日志文件。

2、系统退出时调用一次反初始化

```
void  SDTerminalMulticast_Enviroment_Free()
```

3、创建 SDK 对象

```
void*  SDTerminalMulticast_Create();
```

4、销毁 SDK 对象

```
void  SDTerminalMulticast_Delete(void** ppTerminal);
```

参数:

@ ppTerminal, 模块指针的指针

说明: 使用者应该做好与其他 API 之间的互斥保护

5、准备会话

```
int  SDTerminalMulticast_Online(
    void* pTerminal,
    TERMINAL_TYPE_MULTICAST eUserType,
    const char* strLocalIP,
    const char* strMultiIp,
    unsigned short shMultiPort
    );
```

参数:

@pTerminal, 模块指针

@eUserType: 客户端类型, 有组播发送端与组播接收端两种。

@strLocalIp, 组播绑定的本地 IP 地址, 当设置为 NULL 时, 内部将使用 INADDR_ANY, 交由操作系统选择一个网卡 IP。建议指定 IP 以防多 IP 时系统选择错误。

@strMultiIp, 组播 IP 地址。

@shMultiPort, 组播端口号, 将使用本端口发送音频数据 使用本端口+2 发送视频数据。

返回值:

返回大于等于 0 表示成功, 返回负数则为失败, 负数值为其错误码。

6、结束会话

```
void  SDTerminalMulticast_Offline(void* pTerminal);
```

参数:

@pTerminal，模块指针

返回值: 无

7、发送视频数据

```
void  SDTerminalMulticast_SendVideoData(void* pTerminal, unsigned char *byBuf, unsigned int unLen, unsigned int unDts, BOOL bIsHevc);
```

发送视频码流，一次传入**一帧带起始码(0x 00 00 00 01 或 0x00 00 01)**的码流。

参数:

@pTerminal，模块指针

@byBuf，码流存放区。

@unLen，码流长度。

@unDts，SDK 默认使用内部时间戳模式，内部自动产生时间戳，此时本参数传入 0 即可。当用户调用 SDTerminalMulticast_SetUseInternalTimeStamp API 来指定外部时间戳模式时，在此传入外部时间戳。@bIsHevc，当前码流是否为 HEVC (H265) 码流，是则设置为 TRUE，H264 码流设置为 FALSE，请务必按实际情况准确设置。

@ bIsHevc:传入码流是否为 HEVC，H264 时传入 FALSE，请务必与实际情况相符。

返回值: 无

8、发送音频数据

```
void  SDTerminalMulticast_SendAudioData(void* pTerminal, unsigned char *byBuf, unsigned int unLen, unsigned int unDts);
```

向请求的位置发送音频码流，一次传一帧 **ADTS** 码流。内部将校验 ADTS 头合法性。

参数:

@pTerminal，模块指针

@byBuf，码流存放区。

@unLen，码流长度。

@unDts，SDK 默认使用内部时间戳模式，内部自动产生时间戳，此时本参数传入 0 即可。当用户调用 SDTerminalMulticast_SetUseInternalTimeStamp API 来指定外部时间戳模式时，在此传入外部时间戳。

返回值： 无

9、设置音视频传输参数

```
void SDTerminalMulticast_SetTransParams(void* pTerminal, unsigned int unJitterBuffDelay,
FEC_REDUN_TYPE_MULTICAST eFecRedunMethod, unsigned int unFecRedunRatio,
unsigned int unFecMinGroupSize, unsigned int unFecMaxGroupSize,
BOOL bEnableNack,
unsigned short usNackPort);
```

参数：

@pTerminal，模块指针

@unJitterBuffDelay，本客户端接收码流时的内部缓存时间（毫秒），范围 0~600。设置为 0 时，将关闭内部接收 JitterBuff 功能，此时可以获得最低延时。

@eFecRedunMethod，为上行 FEC 冗余度方法，包括 AUTO_REDUN 自动冗余度、FIX_REDUN 固定冗余度。自动冗余度将根据网络情况自行调整冗余。固定冗余度则全程使用固定值。

@unFecRedunRatio，上行冗余比率，比如设置为 30，则表示使用 30%冗余。自动冗余度时以该冗余度作为基础值，根据网络情况调整。

@unFecMinGroupSize，为上行 FEC 分组的下限值，建议设置为 16。

@unFecMaxGroupSize，为上行FEC分组的上限值，根据终端CPU能力而定，最大不超过72，越大FEC所消耗的CPU越高，抗丢包能力也越强，建议性能足够的设备上设置为64。

@bEnableNack，是否启用组播 NACK 功能，当设置为 TRUE 时启用。收发两端均开启时生效。

@usNackPort，当开启 NACK 功能时，组播发送端用于 NACK 信令、媒体传输的单播端口，收发双方保持设置一致时生效。

返回值： 无

说明：注意，本函数需在 Online 之前调用。

10、获取当前 SDK 版本信息

```
UINT SDTerminalP2P_GetVersion(void* pTerminal);
```

参数：

@pTerminal，模块指针

返回值： 获得当前 SDK 的版本信息

11、获取当前丢包率数据

```
void SDTerminalMulticast_GetVideoAudioUpDownLostRatio(void* pTerminal,
float *pfVideoUpLostRatio, float *pfVideoDownLostRatio,
float *pfAudioUpLostRatio, float *pfAudioDownLostRatio);
```

参数：

@pTerminal，模块指针

@pfVideoUpLostRatio，获取视频上行丢包率

@pfVideoDownLostRatio，获取视频下行丢包率

@pfAudioUpLostRatio，获取音频上行丢包率

@pfAudioDownLostRatio，获取音频下行丢包率

返回值： 无

说明： 上述值内部已经乘 100.0 转换为百分比

12、设置时间戳工作机制

```
void  SDTerminalMulticast_SetUseInternalTimeStamp(void* pTerminal,
BOOL bUseInternalTimestamp);
```

@pTerminal，模块指针

@bUseInternalTimestamp，是否采用内部时间戳模式，TRUE-内部，FALSE-外部。默认情况下未调用本 API 时，系统采用内部时间戳模式，此时将在每次调用 Send 接口时自动产生时间戳。当用户需要使用外部时间戳时，需调用本 API 指定 FALSE。

返回值： 无

说明： 本函数需在 Online 之前调用。不调用本函数时，默认使用内部时间戳模式。

13、设置发送端 Smooth 机制

```
void  SDTerminalMulticast_SetVideoFrameRateForSmoother(void* pTerminal,
unsigned int unFrameRate);
```

@pTerminal，模块指针

@unFrameRate，设置视频帧率信息，该值将作为内部发送时 Smoother 处理的参考。注意该帧率要符合实际帧率,可以高于实际帧率，但不能低于实际帧率，否则将导致发送速度不足，触发内部自行关闭 Smooth 功能。

返回值： 无

说明： 本函数需在 Online 之前调用。不调用本函数时，默认关闭 smooth 处理。

三、回调输出相关 API 接口

接收的远端音视频数据均通过回调函数的方式通知外层，SDK 提供了相关的回调函数设置接口。

1、设置视频数据接收回调

```
void SDTerminalMulticast_SetRecvRemoteVideoCallback(void* pTerminal,
RecvMulticastRemoteVideoFunc pfRecvRemoteVideoCallback, void* pObject)
```

参数：

@pTerminal，模块指针

@pfRecvRemoteVideoCallback,回调函数指针

@pObject，透传指针，将透传给回调函数。

RecvMulticastRemoteVideoFunc 的定义如下：

```
void (*RecvMulticastRemoteVideoFunc)(void* pObject, unsigned char* data, unsigned int unLen,
unsigned int unPTS, VideoFrameInforMulticast* pFrameInfo);
```

其中，VideoFrameInforMulticas 提供了当前帧以及当前流的重要信息：

```
typedef struct VideoFrameInforMulticast
{
    unsigned int unWidth;
    unsigned int unHeight;
    unsigned int unFps;
    BOOL bPacketLost;
    BOOL bKeyFrame;
    BOOL bInfoUpdated;
    BOOL bIsHevc;
```

```
    unsigned char byVps[512];
    unsigned int unVpsSize;
    unsigned char bySps[512];
    unsigned int unSpsSize;
    unsigned char byPps[512];
    unsigned int unPpsSize;
} VideoFrameInforMulticast;
```

模块内部会获得当前码流的宽、高、帧率、VPS（HEVC 时）、SPS、PPS 告知外层，当其中某些参数发生变更时将置位 bInfoUpdated 以通知外层。

bPacketLost 与 bKeyFrame 变量可用于外层实现丢帧冻结机制，bPacketLost 表示当前帧是否接收完整，若网络丢包且 FEC 未能恢复时，该标志将置位。bKeyFrame 表示当前帧是否为 IDR 关键帧。默认情况下内部已开启丢包冻结机制，外层可忽略 bPacketLost 与 bKeyFrame。

当没有丢包发生时，本函数的输出与对方调用 SDTerminalP2P_SendVideoData 函数的输入完全一致。

返回值： 无

说明： SDK 内部将在独立于网络接收线程之外的线程中调用本接口，所以外层可以将相对耗时的操作（比如解码）放置在此回调中。

2、设置音频数据接收回调

```
void  SDTerminalMulticast_SetRecvRemoteAudioCallback(void* pTerminal,
RecvMulticastRemoteAudioFunc pfRecvRemoteAudioCallback, void* pObject);
```

参数：

@pTerminal，模块指针

@pfRecvRemoteVideoCallback,回调函数指针

@pObject，透传指针，将透传给回调函数。

RecvMulticastRemoteAudioFunc 的定义如下：

```
void (*RecvMulticastRemoteAudioFunc)(void* pObject, unsigned char* data, unsigned int unLen,
unsigned int unPTS, AudioFrameInforMulticast* pFrameInfo);
```

其中，AudioFrameInforMulticast 提供了当前帧以及当前流的重要信息：

```
typedef struct AudioFrameInforMulticast
{
```

```
    unsigned int unCodecType;
    unsigned int unSampleRate;
    unsigned int unChannelNum;
    unsigned int unFrameNo;
    BOOL bInfoUpdated;
} AudioFrameInforMulticast;
```

音频帧为 ADTS 格式，其每个包头部均附带了采样率、通道数、编码格式（目前固定 0-AAC）等信息。

返回值： 无

说明： SDK 内部是在独立于网络接收线程之外的线程中调用本接口，所以外层可以将相对耗时的操作（比如解码）放置在此回调中。