瑜媛的专栏

不集小流，无以成江海！

RTP协议之Header结构解析

分类： 网络传输协议 流媒体技术 2013-08-09 09:28 1343人阅读 评论(0) 收藏 举报

RTP 传输协议 RC CSRC

实时传输协议 RTP，R 时特性的端对端数据传输服务，传输的数据如：交互式的音频和视频。那些服务包括有效载荷类型定义，序列号，时间戳和传输监测控制。应用程序在 UDP 上运行 RTP 来使用它的多路技术和 checksum 服务。2 种协议都提供传输协议的部分功能。不过,RTP 可能被其他适当的下层网络和传输协议使用。如

果下层网络支持，RTP 支持数据使用多播分发机制转发到多个目的地。

注意 RTP 本身没有提供任何的机制来确保实时的传输或其他的服务质量保证，而是由低层的服务来完成。它不保证传输或防止乱序传输,它不假定下层网络是否可靠,是否按顺序传送数据包。RTP 包含的序列号允许接受方重构发送方的数据包顺序，但序列号也用来确定一个数据包的正确位置,例如,在视频解码的时候不用按顺序的对数据包进行解码。

但是 RTP 原先的设计是用来满足多参与者的多媒体会议的需要，它没有限定于专门的应用。连续数据的储存，交互分布式仿真，动态标记，以及控制和测量应用程序也可能会适合使用 RTP。

1、RTP 固定头结构

RTP 固定头结构如图2所示。

0								1								2								3							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
V			P	X		CC		M		PT						Sequence Number															
TimeStamp																															
Synchronization Source (SSRC) Identifier																															
Contributing Source (CSRC) Identifier List																															

图1 RTP固定头格式

前 12 个字节出现在每个 RTP 包中，仅仅在被混合器插入时，才出现 CSRC 识别符列表。各个域的含义如下所示：

（1）版本(V)：2 比特，此域定义了 RTP 的版本。此协议定义的版本是 2。(值 1 被 RTP 草案版本使用,值 0 用在最初"vat"语音工具使用的协议中。)

（2）填充(P)：1 比特，若填料比特被设置,则此包包含一到多个附加在末端的填充比特,填充比特不算作负载的一部分。填充的最后一个字节指明可以忽略多少个填充比特。填充可能用于某些具有固定长度的加密算法，或者用于在底层数据单元中传输多个 RTP 包。

（3）扩展(X)：1 比特，若设置扩展比特,固定头(仅)后面跟随一个头扩展。

（4）CSRC 计数(CC)：4 比特，CSRC 计数包含了跟在固定头后面 CSRC 识别符的数目。

（5）标志(M)：1 比特，标志的解释由具体协议规定。它用来允许在比特流中标记重要的事件,如帧边界。

（6）负载类型(PT)：7 比特，此域定义了负载的格式，由具体应用决定其解释，协议可以规定负载类型码和负载格式之间一个默认的匹配。其他的负载类型码可以通过非 RTP 方法动态定义。RTP发送端在任意给定时间发出一个单独的 RTP 负载类型；此域不用来复用不同的媒体流。

（7）序列号(sequence number)：16 比特，每发送一个 RTP 数据包,序列号加 1，接收端可以据此检测丢包和重建包序列。序列号的初始值是随机的(不可预测),以使即便在源本身不加密时(有时包要通过翻译器,它会这样做)，对加密算法泛知的普通文本攻击也会更加困难。

（8）时间戳(timestamp)： 32 比特，时间戳反映了 RTP 数据包中第一个字节的采样时间。时钟频率依赖于负载数据格式,并在描述文件(profile)中进行描述。也可以通过 RTP 方法对负载格式动态描述。

如果 RTP 包是周期性产生的，那么将使用由采样时钟决定的名义上的采样时刻，而不是读取系统时间。例如,对

一个固定速率的音频，采样时钟将在每个周期内增加 1。如果一个音频从输入设备中读取含有 160 个采样周期的块，那么对每个块,时间戳的值增加 160。时间戳的初始值应当是随机的，就像序号一样。几个连续的 RTP 包如果是同时产生的。如:属于同一个视频帧的 RTP 包,将有相同的序列号。

不同媒体流的 RTP 时间戳可能以不同的速率增长。而且会有独立的随机偏移量。因此,虽然这些时间戳足以重构一个单独的流的时间，但直接比较不同的媒体流的时间戳不能进行同步。对于每一个媒体，我们把与采样时刻相关联的 RTP 时间戳与来自于参考时钟上的时间戳(NTP)相关联。因此参考时钟的时间戳就是数据的采样时间。(即:RTP 时间戳可用来实现不同媒体流的同步，NTP 时间戳解决了 RTP 时间戳有随机偏移量的问题。)参考时钟用于同步所有媒体的共同时间。这一时间戳对(RTP 时间戳和 NTP 时间戳)，用于判断 RTP 时间戳和 NTP 时间戳的对应关系,以进行媒体流的同步。它们不是在每一个数据包中都被发送，而在发送速率更低的 RTCP 的 SR(发送者报告)中。

如果传输的数据是存贮好的，而不是实时采样得到的,那么会使用从参考时钟得到的虚的表示时间线(virtual presentation timeline)。以确定存贮数据中的每个媒体下一帧或下一个单元应该呈现的时间。此种情况下 RTP 时间戳反映了每一个单元应当回放的时间。真正的回放将由接收者决定。

（9）SSRC：32 比特，用以识别同步源。标识符被随机生成，以使在同一个 RTP 会话期中没有任何两个同步源有相同的 SSRC 识别符。尽管多个源选择同一个 SSRC 识别符的概率很低，所有 RTP 实现工具都必须准备检测和解决冲突。若一个源改变本身的源传输地址，必须选择新的SSRC 识别符，以避免被当作一个环路源。RTP 包流的源，用 RTP 报头中 32 位数值的SSRC 标识符进行标识，使其不依赖于网络地址。一个同步源的所有包构成了相同计时和序列号空间的一部分，这样接收方就可以把一个同步源的包放在一起，来进行重放。举些同步源的例子，像来自同一信号源的包流的发送方，如麦克风、摄影机、RTP 混频器就是同步源。一个同步源可能随着时间变化而改变其数据格式，如音频编码。SSRC 标识符是一个随机选取的值，它在特定的 RTP 会话中是全局唯一(globally unique)的。参与者并不需要在一个多媒体会议的所有 RTP 会话中，使用相同的 SSRC 标识符；SSRC 标识符的绑定通过RTCP。如果参与者在一个 RTP 会话中生成了多个流，例如来自多个摄影机，则每个摄影机都必须标识成单独的同步源。

（10）CSRC 列表：0 到 15 项，每项 32 比特，CSRC 列表识别在此包中负载的所有贡献源。识别符的数目在 CC 域中给定。若有贡献源多于 15 个，仅识别 15 个。CSRC 识别符由混合器插入，并列出所有贡献源的 SSRC 识别符。例如语音包，混合产生新包的所有源的 SSRC 标识符都被列出，以在接收端处正确指示参与者。

若一个 RTP 包流的源，对由 RTP 混频器生成的组合流起了作用，则它就是一个作用源。对特定包的生成起作用的源，其 SSRC 标识符组成的列表，被混频器插入到包的 RTP 报头中。这个列表叫做 CSRC 表。相关应用的例子如，在音频会议中，混频器向所有的说话人(talker)指出，谁的话语(speech)将被组合到即将发出的包中，即便所有的包都包含在同一个(混频器的)SSRC 标识符中，也可让听者(接收者)可以清楚谁是当前说话人。

2、RTP扩展头结构

RTP 提供扩展机制以允许实现个性化:某些新的与负载格式独立的功能要求的附加信息在RTP 数据包头中传输。设计此方法可以使其它没有扩展的交互忽略此头扩展。RTP扩展头格式如图2所示。

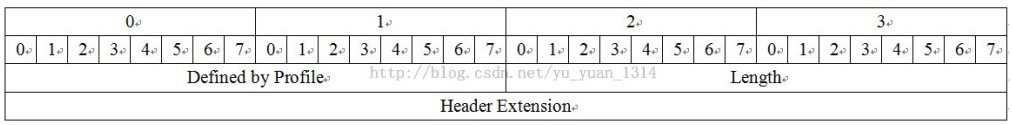

图2 RTP扩展头格式

若 RTP 固定头中的扩展比特位置 1，则一个长度可变的头扩展部分被加到 RTP 固定头之后。头扩展包含 16 比特的长度域，指示扩展项中 32 比特字的个数，不包括 4 个字节扩展头(因此零是有效值)。RTP 固定头之后只允许有一个头扩展。为允许多个互操作实现独立生成不同的头扩展，或某种特定实现有多种不同的头扩展,扩展项的前 16 比特用以识别标识符或参数。这 16 比特的格式由具体实现的上层协议定义。基本的 RTP 说明并不定义任何头扩展本身。

3、RTP包解析

RTP包解析示例如下所示：

```cpp
static int parsingRTPPacket(uint8_t *data, size_t size) {
    if (size < 12) {
        //Too short to be a valid RTP header.
        return -1;
    }

    if ((data[0] >> 6) != 2) {
        //Currently, the version is 2, if is not 2, unsupported.
        return -1;
```

```
    }

    if (data[0] & 0x20) {
        // Padding present.
        size_t paddingLength = data[size - 1];
        if (paddingLength + 12 > size) {
            return -1;
        }
        size -= paddingLength;
    }

    int numCSRCs = data[0] & 0x0f;
    size_t payloadOffset = 12 + 4 * numCSRCs;

    if (size < payloadOffset) {
        // Not enough data to fit the basic header and all the CSRC entries.
        return -1;
    }

    if (data[0] & 0x10) {
        // Header extension present.
        if (size < payloadOffset + 4) {
            // Not enough data to fit the basic header, all CSRC entries and the first 4 bytes
            return -1;
        }

        const uint8_t *extensionData = &data[payloadOffset];
        size_t extensionLength = 4 * (extensionData[2] << 8 | extensionData[3]);

        if (size < payloadOffset + 4 + extensionLength) {
            return -1;
        }
        payloadOffset += (4 + extensionLength);
    }

    uint32_t rtpTime = data[4] << 24 | data[5] << 16 | data[6] << 8 | data[7];
    uint32_t srcId = data[8] << 24 | data[9] << 16 | data[10] << 8 | data[11];
    uint32_t seqNum = data[2] << 8 | data[3];

    return 0;
}
```

上一篇 Darwin Streaming Server搭建RTSP服务器

下一篇 标准C++中Const的详细用法总结