

ISMA
INTERNET STREAMING MEDIA ALLIANCE

Document Number **TD00096**

October 2006

TITLE: Fast channel changing in RTP
AUTHOR: DW Singer, N Färber, Y Fisher, JF Fleury, ISMA TC
AREA / Task Force: IPTV
Status: External Review Document

It is ISMA policy to provide an opportunity for non-ISMA members to review and comment on technical specifications that are being considered for formal adoption by the ISMA membership.

This is NOT a final, approved ISMA Specification.

Table of Contents

| | | |
|-----|--|----|
| 1 | Introduction..... | 3 |
| 2 | Problem Statement | 3 |
| 2.1 | RTSP negotiation..... | 3 |
| 2.2 | IGMP session joining..... | 3 |
| 2.3 | RAP acquisition..... | 4 |
| 2.4 | Client buffering | 4 |
| 2.5 | Synchronization between streams (RTCP sender report) | 4 |
| 2.6 | Encryption keys acquisition | 5 |
| 2.7 | Processing delays..... | 5 |
| 3 | Solutions | 5 |
| 3.1 | In-band Synchronization..... | 5 |
| 3.2 | Server buffering of content | 5 |
| 3.3 | Multicast Configuration | 7 |
| 3.4 | Predictive Tuning (improvement) | 7 |
| 3.5 | RTSP Switching | 7 |
| 3.6 | De-jitter buffering..... | 8 |
| 3.7 | Low resolution stream | 8 |
| 3.8 | End-system load minimization..... | 8 |
| 4 | Recommendations | 8 |
| 5 | Architecture..... | 9 |
| 5.1 | Encoder | 9 |
| 5.2 | Server | 9 |
| 5.3 | Client..... | 10 |

1 Introduction

This paper looks at some of the delays in ‘tune-in’ acquisition of live streams and proposes some techniques of general applicability to minimize the delay perceived by the user. It constitutes some of the preliminary work to establish one or more specifications for fast RTP tune-in.

Though this paper, and the specifications, will detail approaches to minimizing delay, the determination of what constitutes an acceptable delay, and hence the specification of which of these techniques are used, and their variable parameters (e.g. frequency of random access points) is expected to be left to a deployment, unless they impact interoperability.

2 Problem Statement

The delay between the initiation of tune-in and initiation of rendering consists of:

- RTSP negotiation (if requesting streams or stream data)
- IGMP session joining (if tuning to a multicast session)
- Network latency
- Video and audio random access points (RAP) acquisition
- Client stream buffering (to de-jitter reception, to allow time for re-transmission, or to handle video packet re-ordering)
- Synchronization between streams (RTCP sender report)
- Encryption key acquisition
- End-system delays, such as processing delays

Consideration of these indicates that the expected dominant causes of delay would be (a) de-jitter buffering, especially when traffic smoothing is used and (b) acquisition of a video random access point (RAP). Verification experiments should be performed in typical network environments to validate this and acquire figures on the various contributions to delay. Some of these delays can be mitigated, while others, such as network latency, cannot. It should be emphasized that many of the problems are not specific to RTP but arise from coding or other problems. The following sections discuss these delays in more detail.

2.1 RTSP negotiation

RTSP is a protocol that can be used to initiate tune-in to a multicast stream or to request a unicast stream from a server. When it is used, it can take several packet round-trips of request/response before the RTSP server sends the first media packet. In some situations, this can be significant.

2.2 IGMP session joining

If the content is multicast on the network, then there are typically no processes operating at the RTP level between the source and the client. Multicast is an efficient distribution protocol for live streams as each network link needs carry only one copy

of each stream. Routers in the network replicate streams from inputs onto the output links which have one or more clients. Clients indicate their interest in the multicast by sending a special control packet (IGMP), which is intercepted by the router. The routers in turn refer to each other to find, and forward, the packet stream.

This find-and-forward process (called multicast ‘join’) can, in some circumstances, take some time.

2.3 RAP acquisition

Video is classically ‘differentially coded’. Very few coded frames contain all the information needed to create a complete set of decoded pixels. Instead, frames are coded as differences from one or more other frames.

In order to handle both recovery from loss and tune-in to live streams, ‘independently decodable’ frames or I-frames can be sent periodically. Decoder refresh frames or IDR-frames are I-frames that have the additional property that they mark a division of the sequence; frames displayed after the decoder refresh do not depend on frames before it. They are true Random Access Points (RAPs).

Because these frames are so much larger (in bytes) than differentially coded frames, they are sent rarely in order to keep the bit-rate low. A delay occurs when tuning into a new stream, since the decoder must wait for a RAP before it can start displaying video. In addition, their size often means that traffic smoothing causes adjustment of their send time and that of adjacent packets.

2.4 Client buffering

If there is deviation in the arrival time of packets from the relative timing that they would have if they were to arrive just-in-time to be played, then a de-jitter buffer is needed if we are to avoid under-run (starvation). This jitter has two causes: deliberately introduced jitter from the source, for traffic smoothing, and network-introduced jitter (e.g. caused by cross-traffic in network equipment). The source-introduced jitter tends to occur most, or even exclusively, in video, where the variation in coded frame size (in bytes) can be large (e.g. I-frames can be many times as large as B frames).

A buffer is also needed if there is to be time to perform re-transmissions.

A delay occurs during the time the client fills its buffer before starting to render.

2.5 Synchronization between streams (RTCP sender report)

RTP streams have ‘free floating’ timestamps – they have arbitrary origins (and indeed, usually different streams have different tick-rates). In order to synchronize audio and video, their time stamps have to be related. Associated with each stream are periodic RTCP sender reports, which associate the RTP timestamps in the stream with a common clock at the transmitter (usually the time-of-day clock, but actually any clock is permitted as long as it is common to all streams).

Until at least one RTCP sender report has been acquired for each stream, the streams cannot be played in synchronization. The only exception to this is that the RTSP control protocol has provision for sending the initial synchronization information at the beginning of a play interval. However, RTSP is not used in all situations.

Thus, the frequency and timing of RTCP reports often contribute to the delay before audio and video are rendered, not just to their synchronization, because many clients will not render anything before synchronization has been established.

2.6 Encryption keys acquisition

If the content is encrypted, then the decryption keys must be acquired. Generally it is not desirable to acquire keys too far ahead of need (either for the future of the current stream, or for other potential streams). Since this problem is both independent of RTP, and dependent on the key-exchange method used, it is not discussed here.

2.7 Processing delays

Processing delays can occur in the terminal at a number of layers – network, RTP, codec, and so on. In general, this is a trade-off between terminal resources (memory, processor speed) and cost. However, there are recommendations that can be made to minimize the processing load on the end-system, and hence the delay.

3 Solutions

3.1 In-band Synchronization

The fact that RTP synchronization is usually provided (and must be provided) in RTCP does not preclude its provision by other, additional, means. Indeed, one has already been noted – its provision in the RTSP protocol.

Using the header extension defined in <http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-hdext-05.txt>, it is possible to provide the time-mapping in selected RTP packets, in-band. This would take the form of four extra 32-bit words in those packets. (One word indicates that the header extension is in use and the length of all the extensions, then there is a one-byte tag and length for this extension, 8 bytes of NTP timestamp, and, alas, 3 bytes of padding.)

This would typically only need to be sent on those packets where the decoding can start – RAP frames in video – or their temporally aligned packets in other streams. These packets are already quite large, as noted, and this overhead is small in comparison. The temporally-aligned audio packets may be small, and the extension may be a perceptible enlargement of these packets; however, the frequency of random access points in video should be low enough that overall this is still an acceptable overhead.

Implementing this scheme eliminates the delay caused by waiting for the RTCP sender reports.

3.2 Server buffering of content

If a streaming server is being used for the final distribution of content, then it could buffer the stream so that it is able to supply to each client, as it tunes in, a decoder refresh point and its temporally aligned audio packet. Servers doing packet reflection already ‘pace’ their output packet stream, not sending N copies of each incoming packet to their N clients simultaneously, but spreading them out in time. If they buffer each decoder refresh interval, providing the beginning of the most recent one, to each client as they tune-in, this will have a similar stochastic spreading effect.

In order to do this, the packet stream is ideally marked in a codec-independent fashion. Indeed, it has long been an awkward nature of RTP that random-access

points (RAPs) – for tune-in or error-recovery – are not marked at the RTP level. The general header extension mechanism can again be used, marking packets as RAPs, or as aligned with the RAP in a synchronized stream (such as a parallel audio stream). Such marking can also label frames which are independently decidable even if not RAPs, or as ‘leaves’ in the dependency graph (i.e. no other frame depends on them).

If buffering every frame of the stream, for the intervals desired, is unacceptable for memory reasons, then the server could choose to buffer only video frames that have dependencies, or indeed only I frames.

There are two ways to use this buffering. The client may use the unicast content from the server only during the tune-in period; once it has acquired the multicast, and the packet sequence numbers overlap between the unicast and multicast feeds, then the unicast can be dropped (the streams splice perfectly). Alternatively, it may be desirable to use client-server unicast for the duration of the stream, as this enables retransmission and client-server buffer management.

If the server is performing application-level replication (as opposed to IP multicast, which is network-level) then it needs to be placed ‘suitably close’ to the point where the physical transmissions diverge, as it has N copies of the input traffic for N clients, on its output link. This is not necessarily co-located with the router, for example, that supplied the CPE. There are deployment cost trade-offs between many smaller servers located close to the routers, and a larger one deeper in the network, with suitable bandwidth out to the routers that feed the CPE.

The load on those routers varies, of course. If they are replicating multicast, only one packet is received for N client packets transmitted, whereas if an upstream server is replicating, then N unicast packets are received. Unicast routing is typically cheaper than multicast routing, but this does represent a change of load.

The tagging is done using the header extension in RTP, referred to above. If the tags for video are present in an audio stream, then they provide information about the temporally aligned video. The tag has the name `tv.isma.rtp.frametags.video/092006`.

The video tag is a one-byte field with the following values. This tag may be extended in future with extra bytes, which may be ignored if not recognized. It has the following definition:

```
unsigned int(1) reserved = 0;
unsigned int(1) RAP;
unsigned int(2) sample_depends_on;
unsigned int(2) sample_is_depended_on;
unsigned int(2) sample_has_redundancy;
```

RAP takes one of the following two values:

- 0: not known to be a random access point
- 1: Random access point

`sample_depends_on` takes one of the following four values:

- 0: the dependency of this sample is unknown;
- 1: this sample does depend on others (not an I picture);
- 2: this sample does not depend on others (I picture);
- 3: reserved

`sample_is_depended_on` takes one of the following four values:

- 0: the dependency of other samples on this sample is unknown;
- 1: other samples depend on this one (not disposable);
- 2: no other sample depends on this one (disposable);

3: reserved

sample_has_redundancy takes one of the following four values:

0: it is unknown whether there is redundant coding in this sample;

1: there is redundant coding in this sample;

2: there is no redundant coding in this sample;

3: reserved

3.3 Multicast Configuration

We discuss two methods to reduce delays associated with joining a multicast session.

Single-source multicast enables the routers to follow normal unicast routing towards the multicast source. This does not require special multicast configuration and routing tables, and can be faster.

Also, on a managed network it is possible to setup the multicasts so that they normally 'flood' the network down to the last switching node on the network (for instance the DSLAM). The IGMP join is thus limited to the last mile and session joining is faster.

Some systems use SAP for program acquisition; SAP is a protocol in which program information, in SDP format, is multicast at a very low rate. Since the rate is low, SAP listeners generally have to be 'permanently on' (listening in the background). The delay that would be caused by waiting for an announcement without this background listener could be large, and is completely avoided by the background listening and cache.

3.4 Predictive Tuning (improvement)

One of the simplest techniques for reducing channel switch times is to predict what channel the user will next ask for, and tune that in early. If there is enough bandwidth, holding open channels that this user often watches, or the 'next' and 'previous' channels (for the channel surfer).

However, this consumes the bandwidth for multiple streams for each client, and this may use excessive bandwidth. It also does not optimize all stream switches, and so is at best a partial solution.

For deployments based on VDSL (25-50 Mbps) and FTTH there may be enough bandwidth available and the system is designed to handle several streams to each household anyway. In any case, predictive tuning is a technique that can be implemented at the client, most probably without the need for additional protocols.

3.5 RTSP Switching

The following 3GPP SA4 contribution, <http://ietfreport.isoc.org/idref/draft-einarsson-mmusic-rtsp-macuri/>, suggests techniques for reducing delays associated with RTSP handshaking. This proposes that if the channels concerned are indeed channels in a family, then they can presumably share codecs and codec settings, screen size, audio channel count etc., and then the usual three round trips to tear-down and setup a channel over RTSP can be reduced.

Clearly round-trip delays by themselves can be optimized (e.g. the SETUPS can be pipelined, or a new RTSP command used to switch video and audio streams on existing ports).

3.6 De-jitter buffering

Unless the content is encoded at a constant bit-rate, or its peak bit-rate is below the channel rate – whereupon no source-side traffic smoothing need occur – there is going to be jitter in the packet arrival times, caused by traffic smoothing. In a managed network the network-induced jitter can be minimized.

One technique for the client is to notice the first I-frame to arrive, and to display it, even before ‘full playback’ has started. This presents a picture ‘early’ on the screen.

A more systematic approach is to use a protocol for managing re-transmissions, bandwidth usage, and client-side buffer space, such as the one defined by 3GPP 26.234. If the content bandwidth is lower than the network bandwidth, these protocols can build the client-side buffer faster than real-time, so a buffer that takes 3 seconds to play out can be built in a second or less.

In addition, even if the ‘ideal’ buffer is large, it may be worth risking underflow for a short period, deliberately playing the content slightly slowly so that the buffer builds towards a safer value.

In 3GPP 26.234 <<http://www.3gpp.org/ftp/Specs/html-info/26234.htm>> (currently version 6.8), section 6.2.3.3 defines how to perform RTP retransmission, and section 10 details a client-server buffer management and link-rate adaptation protocol. Used together, a streaming server and client can both build the de-jitter at faster than the link-rate, and then use that buffered period as a window in which re-transmissions can be requested.

3.7 Low resolution stream

In order to limit the number of equipments on the network, a second low resolution stream, conveying mainly I frames or with a small GOP (IPP for instance) can be sent in parallel when the user changes channel.

This stream will have a small bandwidth, but will deliver an I frame within at most 200ms. This stream will be up-scaled to fit in the targeted resolution and used until an I frame arrives on the full resolution / long GOP stream. When the full I frame has arrived, the low resolution stream is stopped.

The solution is scalable as no new servers are required on the network; it just needs other encoders to generate this low-resolution stream at the headend. And this stream can also be already present on the network to be used for PiP, portable terminals, or for a selection mosaic or channel-guide preview.

3.8 End-system load minimization

The impact on end-system load of various aspects of RTP and codec management should be investigated, with a view to minimizing the load, and hence processing delay. It may be that the load can be reduced, for example, by using fewer network ports. (There is a draft before the IETF now on that subject.)

4 Recommendations

1. Use the RTP extension as defined above for stream labeling of frame types, dependencies etc., that is codec-independent, and also can be applied to other streams in the same content (e.g. “this audio frame happens at the same time as a video I-frame”).

2. Use the extension defined above for in-band periodic transmission of the RTP-NTP (sender report) mapping.
3. Look at client-server buffer management protocols, particularly 3GPP 26.234 release 6, with a view to adopting a 'fast buffer build' approach. Issues to analyze include network equipment load, cost, and complexity.
4. Define how streaming servers doing live stream replication (a.k.a. reflection) can use the meta-data defined above to provide a better experience. Issues to analyze include scalability, network equipment load, cost, and complexity.
5. Define how to manage the low resolution streams from the STB perspective
6. Define the means to get the encryption keys for a live stream.
7. Define that clients acquire a program by performing an RTSP transaction with a server, even if the reply is sometimes the simple instruction 'join the indicated multicast'. This enables the operator to optimize where, and how, each stream is replicated, and the extent to which it is optimized. If clients assume that all streams can be joined as multicasts, then both flexibility, and many optimization opportunities, are precluded.

5 Architecture

5.1 Encoder

1. Use the RTP header extension to insert NTP-RTP time mappings on at least the RAP points in video, and possibly more (e.g. all the I frames). Provide the same mapping also in the audio stream, with the same frequency, preferably in packets that are aligned in time with the tagged packets in the video. This permits the clients to acquire synchronization rapidly, and at the significant points in the stream.
2. Use the RTP header extension to tag the streams with dependency information. This permits streaming servers to find the RAP points, or I frames, and optionally discard frames which have nothing dependent upon them. The cross-tagging enables the matching section of audio to be buffered also.
3. Optionally, use the transmission offset RTP header extension to enable the client to recover the source clock more rapidly.

5.2 Server

1. Build a table of streams that should be buffered (e.g. the commonly used streams). Buffer also any stream that is in use. Buffer at least one entire interval from the most recent RAP point, but consider buffering as much as the client would want as a de-jitter buffer.
2. Even if the program is available multicast, provide the initial buffer fill to the client using unicast. Continue providing packets long enough that the packets from the server and the packets from the multicast must have an overlap in sequence numbers, or until the client indicates that it has the overlap (e.g. by some kind of RTSP transaction).

5.3 Client

1. Always tune-in to streams following an RTSP URL. If the referenced server is the same as the server for the currently open stream, consider using fast RTSP stream-switching. In any case, pipeline setup requests to minimize round-trip delays. This provides an opportunity for (a) choice by the server as to whether the content is unicast or multicast (b) indicating to the server which protocol options are in use (c) re-direction etc., as needed.
2. If the response to 'describe' indicates that the stream is available multicast, nonetheless indicate the port setup to the server. This allows the server to supply the buffered material, to the same ports, unicast. If this happens, indicate to the server when the unicast server 'pre-fill' and the multicast packets are overlapping, and the unicast feed can stop.
3. Indicate to the server willingness to take part in client-server buffer management, and in client-server retransmission. These enable fast buffer pre-fill, and error recovery.
4. Look for in-stream sync information and use it if present.