Seamless Media Adaptation with Simultaneous Processing Chains.

Darren Carlson
NEC Europe Ltd., Network Laboratories
Adenauerplatz 6
69115 Heidelberg, Germany
Darren.Carlson@ccrle.nec.de

Andreas Schrader
NEC Europe Ltd., Network Laboratories
Adenauerplatz 6
69115 Heidelberg, Germany
Andreas.Schrader@ccrle.nec.de

ABSTRACT

In this paper we investigate a seamless method for the adaptation of streamed media by using simultaneous processing chains. The common approach for media adaptation is to deconstruct the current media processing chain and to construct a new media chain afterwards. As a consequence, media frames might be lost and the adaptation cycle needs significant time. In our approach, the new processing chain is built in parallel to the current one and the stream is switched to the new chain as soon as possible. With this mechanism, we can guarantee zero-loss behaviour in the sender, and at the same time, reduce the overall adaptation time significantly. The proposed method is independent of the actual codec and can be applied to both audio and video streams. We present a formal calculation of the possible improvements of our proposed mechanism with respect to information loss and adaptation speed and also report about results obtained in our implementation using the Java Media Framework.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General

General Terms

Algorithms, Performance

1. INTRODUCTION

IP networks are seen as promising technologies for the delivery of advanced multimedia services. However, due to the sensitive nature of real-time multimedia data and the inherent characteristics of current wireless and wired IP networks, high-quality multimedia communication can be problematic and error-prone. New mechanisms for realising Quality-of-Service (QoS) are therefore needed. A large number of mechanisms has already been developed to support service guarantees or traffic differentiation with higher priorities for real-time traffic (e.g. DiffServ [2], IntServ [5]). But these kind of mechanisms are not appropriate in situations of highly fluctuating network parameters due to physical limitations, especially in the case of wireless networks. In order to support end-to-end QoS in such scenarios, adaptive mechanisms in end-systems are required in addition. Recently, a number of approaches for integrated adaptive QoS management systems have been proposed (e.g. [1], [7], etc.)

Very often, it is possible to improve overall streaming quality by changing parameters for audio and video codecs at the sender based on appropriate feedback monitoring. By inserting filter modules or changing the involved compression algorithm, the stream can be tailored to the current network situation. This adaptation process itself, however, also introduces additional quality problems due to the complex and time consuming process of deconstructing and rebuilding media processing chains.

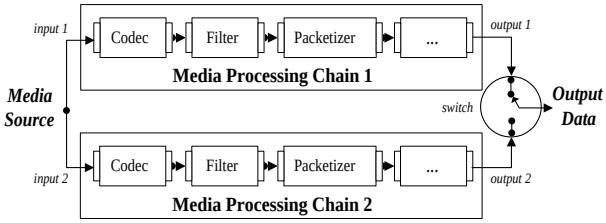

Figure 1: Example Sender Parallel Media Processing Chain.

A media processing chain is often a complicated collection of codecs, packet handlers, media filters, and memory buffers for processing media data (see figure 1 for an example of a sender with two concurrent media processing chains). Adapting such a media processing chain often requires a lengthy process of deconstructing parts of chain or the complete chain, and rebuilding a replacement.

This process is impacted by two types of delay, present within every element of the media processing chain - setup delay and intrinsic delay. Setup delay is defined as the time required by the processing element to initialise its internal data structures, acquire necessary resources and become ready for operation. Intrinsic delay is characterised as the additional time, beyond setup delay, required by a processing element to produce a particular output with a given input. Together, setup and intrinsic delays can introduce

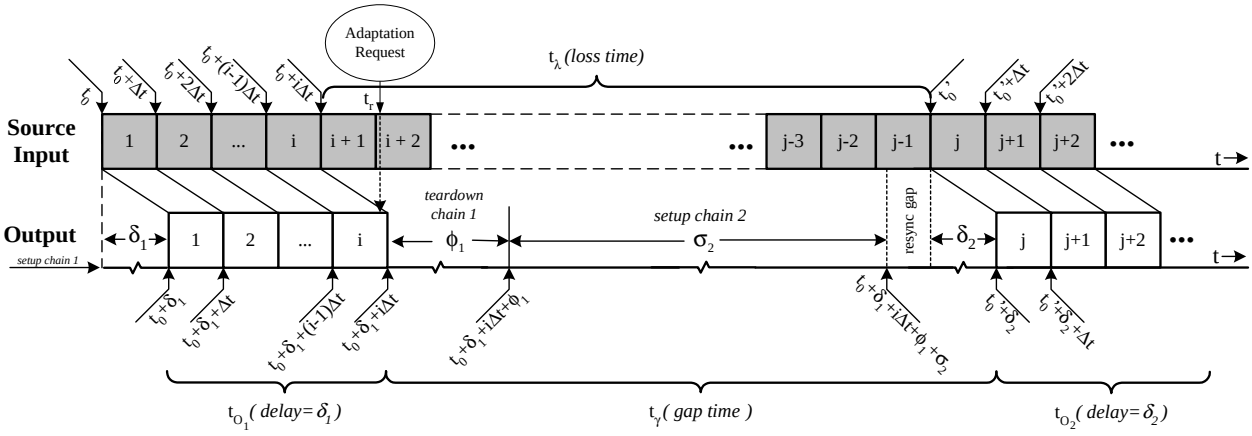

Figure 2: Conventional Media Chain Adaptation through sequential chain building.

additional levels of data loss and extended delays that can degrade quality.

Conventional adaptation mechanisms utilize a simplified method of deconstructing and reconstructing media processing chains without regard for either setup or intrinsic delays. Since these delays can be significant and cause data loss, we propose a new seamless method that avoids these effects.

The remainder of this paper is organized as follows. In section 2, an analytical model for the conventional approach is given as a reference to allow for verifying formally the improvements gained with our proposed seamless method in section 3. In section 4 we demonstrate the improvements for some examples and report from our prototype implementation using the Java Media Framework (JMF, [6]) of Sun Inc. Finally we conclude our paper and present an outlook to future work.

2. CONVENTIONAL APPROACH

In the conventional approach, the current media chain will be deconstructed before the new media processing chain is constructed. During this time, some data will be lost and there will be also a gap in the output stream.

Fig. 2 demonstrates the common mechanism to realize media adaptation. A media source (e.g. camera) is producing raw video frames with framerate $f[\frac{1}{s}]$. The raw data is used as input I in the processing chain C_1. We can consider this input as synchronized data frames of length $\Delta_t = \frac{1}{f}$, which will be available at the processing chain at synchronous times $\{t_0, t_0 + \Delta_t, t_0 + 2 \cdot \Delta_t, \ldots\}$.

The time needed to setup a complete media chain C_i including codecs, filters, buffers, (de-)packetizers and all involved resources is denoted as σ_i. We assume, that the media chain C_1 is already setup at t_0 and frame 1 is the first frame to be processed. The intrinsic delay of the media chain C_i is denoted as δ_i. This is the time needed internally to process the input and create output. The output of the media chain can therefore be generated at the times $\{t_0 + \delta_1, t_0 + \delta_1 + \Delta_t, t_0 + \delta_1 + 2 \cdot \Delta_t, \ldots\}$.

Without loss of generality we assume, that there is a media management system involved (e.g. the MASA QoS Framework [3]), which issues an adaptation request at time t_r. We also assume, that this request is issued during the playout of data packets for frame number i. The playout of all packets of that frame is continued. Afterwards the complete media processing chain is closed. Depending on the involved codecs and packetizers as well as on the media parameters (e.g. video frame rate) the playout process of the last frame will be finished latest at time $t_0 + \delta_1 + i \cdot \Delta_t$. The time necessary to release all the resources of chain C_i is denoted as ϕ_i. This process is finished at time $t_0 + \delta_1 + i \cdot \Delta_t + \phi_1$. Now, the new media processing chain C_2 is constructed, which needs σ_2 seconds to finish. To avoid large buffers and delays, we assume, that the processing of the frames inside the chain cannot be performed faster than realtime, which means that the processing of input media can only be started with the arrival of a complete new input frame and that the next available input frame for the new chain after finishing the setup phase is frame number j. This could lead to a small period of time which is needed to wait for frame number j (indicated as the resync phase in fig. 2).

The new chain starts to consume raw data at time t'_0 and due to the internal delay δ_2, the first output is available at $t'_0 + \delta_2$. During the interval $[t_0 + \delta_1 + i \cdot \Delta_t, t'_0 + \delta_2]$ no output is produced. We denote this time period the gap time t_γ.

The start time of chain C_2 can be calculated as

$$\begin{aligned} t'_0 &= t_0 + \lceil \frac{\delta_1 + i \cdot \Delta_t + \phi_1 + \sigma_2}{\Delta_t} \rceil \cdot \Delta_t \\ &= t_0 + \lceil i + \frac{\delta_1 + \phi_1 + \sigma_2}{\Delta_t} \rceil \cdot \Delta_t \end{aligned} \quad (1)$$

With $t'_0 = t_0 + (j-1) \cdot \Delta_t$ we can determine the first frame to be played out by the new chain:

$$j = i + \lceil \frac{\delta_1 + \phi_1 + \sigma_2}{\Delta_t} \rceil + 1 \quad (2)$$

With $t_\lambda = (j-1) \cdot \Delta_t - i \cdot \Delta_t$ we get the loss time t_λ:

$$t_\lambda = \lceil \frac{\delta_1 + \phi_1 + \sigma_2}{\Delta_t} \rceil \cdot \Delta_t \quad (3)$$

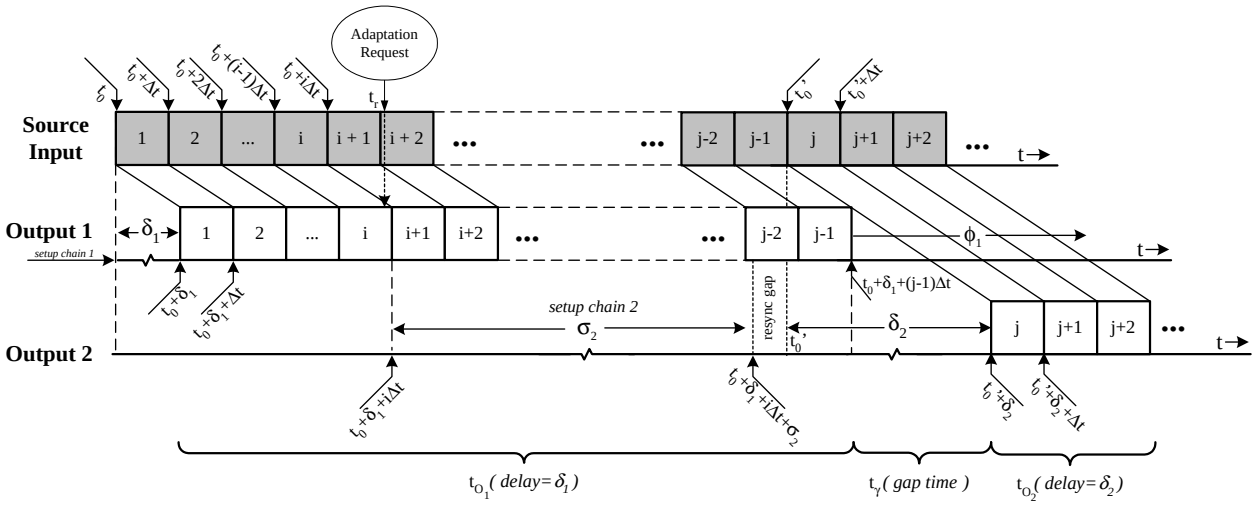

Figure 3: Seamless Media Chain Adaptation through parallel chain building.

We can also determine the number of unprocessed packets during that time period by

$$\lambda = j - 1 - i = \lceil \frac{\delta_1 + \phi_1 + \sigma_2}{\Delta_t} \rceil \quad (4)$$

The gap time is the period, where no output is produced during the adaptation phase:

$$\begin{aligned} t_\gamma &= t'_0 + \delta_2 - (t_0 + \delta_1 + i \cdot \Delta_t) \\ &= \lceil \frac{\delta_1 + \phi_1 + \sigma_2}{\Delta_t} \rceil \cdot \Delta_t + (\delta_2 - \delta_1) \end{aligned} \quad (5)$$

In the conventional case, the time t_α needed to fulfil the adaptation request (which is the time between the request and the playout of the first frame from chain C_2) is exactly the gap time: $t_\alpha = t_\gamma$. Depending on the setup time of chain C_2, the adaptation will not only be slow, but will also produce some lost frames during the adaptation.

3. SEAMLESS APPROACH

Our new seamless approach improves the situation significantly be shifting processing tasks in order to minimize loss and delay of adaptive playout.

Fig. 3 demonstrates the new approach. For the input data, the same argumentation as in the previous subsection applies.

Again, at t_r an adaptation request is issued by the media management system during the playout of data packets for frame number i. But in our seamless approach, the current active chain is not closed immediately and the processing and playout of data frames continues. Instead, the setup of the new media chain C_2 is performed in parallel. The setup phase needs σ_2 seconds. We again denote the first subsequent input frame available after setup as frame j. We also again denote this time event t'_0. The chain now starts to consume raw data and due to the internal delay δ_2, the first output is available at $t'_0 + \delta_2$.

Since the setup of chain C_2 is performed in parallel, chain C_1 produced compressed data for transmission including frame number $j-1$. Since chain C_2 starts with frame number j, actually none of the frames will be lost.

Since $\phi_1 = 0$ the start time of chain C_2 can be calculated as

$$\begin{aligned} t'_0 &= t_0 + \lceil \frac{\delta_1 + i \cdot \Delta_t + \sigma_2}{\Delta_t} \rceil \cdot \Delta_t \\ &= t_0 + \lceil i + \frac{\delta_1 + \sigma_2}{\Delta_t} \rceil \cdot \Delta_t \end{aligned} \quad (6)$$

With $t'_0 = t_0 + (j-1) \cdot \Delta_t$ we can determine the first frame to be played out by the new chain as:

$$j = i + \lceil \frac{\delta_1 + \sigma_2}{\Delta_t} \rceil + 1 \quad (7)$$

The gap time can be determined as follows:

$$\begin{aligned} t_\gamma &= t'_0 + \delta_2 - (t_0 + \delta_1 + (j-1) \cdot \Delta_t) \\ &= \delta_2 - \delta_1 \end{aligned} \quad (8)$$

In the case of $\delta_2 > \delta_1$, a small gap will occur during playout which can be compensated by buffers. These buffers will be used anyhow to compensate jitter. The gap is based on the delay difference between the codecs and is independent from our mechanism. If the intrinsic delay values are equal, the stream can be continously played out. If $\delta_2 < \delta_1$, the playout of the new media chain is available even before the playout of the current media chain. Therefore, it is up to the algorithm to decide, at which frame the new chain will be used. This decision could be based on the actual datarate produced for the certain frame by both chains. One solution could be to switch to the new chain in the moment, the output of C_2 creates a smaller datarate. In both cases, no frames are lost and therefore

$$t_\lambda = 0, \lambda = 0 \quad (9)$$

With the seamless approach, the adaptation time t_α is also shorter, since the new chain is able to playout frames earlier:

$$\begin{aligned} t_\alpha &= t'_0 + \delta_2 - (t_0 + \delta_1 + i \cdot \Delta_t) \\ &= t_0 + \lceil i + \frac{\delta_1 + \sigma_2}{\Delta_t} \rceil \cdot \Delta_t + \delta_2 - t_0 - \delta_1 - i \cdot \Delta_t \\ &= \lceil \frac{\delta_1 + \sigma_2}{\Delta_t} \rceil \cdot \Delta_t + (\delta_2 - \delta_1) \end{aligned} \quad (10)$$

4. RESULTS

By using the analytical model presented in sections 2 and 3, we can show the improvements gained with some example values. If we assume 'perfect' processing chains, which can be instantiated instantaneously, and which have an intrinsic codec delay of $\delta = 0$ms, both conventional and seamless mechanisms are of equal performance. But this is of course an unrealistic scenario. Even for very fast setup and teardown times, e.g. $\sigma_2 = \phi_1 = 50$ms and low delays of $\delta_1 = 50$ms, $\delta_2 = 30$ms, in the conventional case we already have a gap time of $t_\gamma = 180$ms, resulting in 4 lost frames. With our method this can be significantly improved by 89% ($t_\gamma = 20$ms). No frames will be lost at all. For higher delay and setup values, which typically arise in real implementations, the results are even better.

As an example multimedia middleware, we used the Java Media Framework (JMF) of Sun Inc. [6]. JMF provides an comprehensive multimedia API allowing for constructing processing chains. JMF itself is not able to switch between codecs during processing. We have extended the API with adaptation modules realizing our proposed simultaneous method. Due to the space restrictions, we concentrate here on the results for the audio codecs supporting RTP transmission in JMF (i.e. DVI, G.711, G.723, GSM and MP3 (MPEG-2 Layer 3)). In our sender, we have implemented an adaptation cycle going through all possible codec changes, where every 20 seconds an adaptation is performed. The measurements were performed using a *CaptureDataSource* querying audio samples from a microphone.

[ms]	DVI	GSM	G.711	G.723	MP3
DVI	174.4	586.8	667.0	583.0	140.0
GSM	178.2	586.6	600.6	589.0	140.4
G.711	188.2	598.8	678.8	556.6	142.2
G.723	182.4	550.4	671.0	588.8	148.8
MP3	234.2	585.0	654.8	603.0	148.2

Table 1: Gap times t_γ without seamless approach.

Table 1 demonstrates the large gap times, that can occur using the conventional method with average values over 10 runs for each possible adaptation on an Athlon Thunderbird (750MHz, 256MByte) running Windows 2000. Even though the values vary somehow, it can be clearly observed, that the adaptation time strongly depends on the target codec. Even for the fastest codec available (MP3), we observe a gap time of 140ms, resulting in a loss of 2-3 packets on average. Similar experiments with a *FileDataSource* have shown even worse results due to uncontrolled garbage collection and other effects. In the worst case, delays exceeded 3 seconds, leading to significant frame loss. For devices with limited computing power (i.e. mobile devices) acceptable adaptation cycles may not be possible.

With our proposed seamless method we observed significant improvements with respect to both delay and frame loss. The gap time was always smaller than 1ms (measurement accuracy). Packet loss was completely eliminated which was verified utilizing a packet sniffer tool [4]. We must mention nevertheless, that the parallel instantiation of a second processing chain causes a slight increase in jitter values for the last 5-10 packets of the old chain and the first 5-10 packets of the new chain, but this could be easily eliminated with a jitter compensation buffer. The results for video are even better, since the setup times for video codecs are much higher.

5. CONCLUSIONS

In this paper we have demonstrated a seamless adaptation scheme for multimedia transmission. By using simultaneous media processing chains instead of sequential teardown and setup phases, the adaptation process has been significantly improved. We have demonstrated our results with an analytical model as well as with a Java-based implementation. Gap times and adaptation times have been significantly reduced and loss of unprocessed data frames could be avoided completely.

The proposed method could be easily enhanced with additional features. For example, in some cases it might be useful to create more than one parallel chain, e.g. to support hierarchical codecs in powerful machines. The time to switch from the previous to the new chain could also be based on additional parameters, e.g. produced datarate. This could be very useful to allow for settling of involved processing modules (some codecs initially produce a higher datarate). In some cases, it might also be useful to not immediately release the resource of the old chain after the switch, since in future adaptation requests, the old chain might be needed again. Sometimes, the process might even be optimized by re-using parts of the old chain (e.g. the codec) for the new chain and replacing a sub-chain (e.g. filter modules).

Currently we are working on additional implementations of our mechanism using the new Quicktime 6 API and are also planning to support the same process using the Java Multimedia API in the Java 2 Micro Edition on small end devices.

6. REFERENCES

[1] C. Aurrecoechea, A. Campbell, and L. Hauw. A Survey of QoS Architectures. *Multimedia Systems Journal*, 6(3):138–151, May 1998.

[2] A. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *RFC2475: An Architecture for Differentiated Services.* IETF.

[3] D. Carlson, H. Hartenstein, A. Schrader. QoS Orchestration for Mobile Multimedia. In *Proceedings of the First Workshop on Applications and Services in the Wireless Networks, ASW'2001*, July 2001.

[4] GNU. *The Ethereal Packet Sniffer Tool.* http://www.ethereal.com/.

[5] D. C. R. Braden and S. Shenker. *RFC1633: Integrated Services in the Internet Architecture: An Overview.* IETF, June 1994.

[6] Sun. *The Java Media Framework Version 2.0 API.* http://java.sun.com/products/java-media/jmf.

[7] B. Vandalore, R. Jain, S. Fahmy, and S. Dixit. AQuaFWIN: Adaptive QoS Framework for Multimedia in Wireless Networks and its Comparison with other QoS Frameworks. In *Proc. LCN'99*, Oct 1999.