

Demonstrating ATSC ROUTE-DASH Delivery

Waqar Zia
Nomor Research GmbH
Brecherspitzstr. 8, 81549, Munich,
Germany
+49 89 978980 27
zia@nomor.de

Thomas Stockhammer
Qualcomm CDMA Technologies
GmbH
Franziskanerstr. 14, 81669 München,
Germany
+49-6196-9993-108
tsto@qti.qualcomm.com

Kent Walker
Qualcomm Technologies, Inc.
5775 Morehouse Dr., San Diego, CA
9212, USA
+1-858-651-7228
gwalker@qti.qualcomm.com

ABSTRACT

DASH content delivery via ROUTE is being standardized within ATSC and is supported by the interoperability efforts within DASH-IF. The goals achieved by this work include effectively demonstrating the key technological features of the nascent standards on a variety of platforms. These features include demonstration of hybrid services delivery and customized advertisement insertion, etc. At the same time, this demonstration is accomplished using open-source tools, content, and components to help foster harnessing and utilizing the full potential of these technologies in the industry. All of this is supplemented by near-product grade stable and optimized performance demonstration.

CCS Concepts

• Information systems → Information systems → Information systems applications → Multimedia information systems → Multimedia streaming • Information systems → Information systems applications → Mobile information processing systems

Keywords

DASH; ROUTE; Broadcast; Hybrid Services; Demonstration

1. INTRODUCTION

In the recent years, industry has focused to utilize Dynamic Adaptive Streaming over HTTP (DASH) [4] format for broadcast media. The well-known reason is a large set of existing multimedia content in DASH format, and hence it is well suited to serve as a native format for delivery over broadcast. The content can then be played back using well-known DASH clients, minimizing the investment and efforts required to realize the content generation and consumption framework, while at the same time optimizing the overall broadcast system resource usage for multimedia delivery.

Several standardization bodies including European Telecommunications Standards Institute (ETSI) for their DVB standards [3], 3GPP for eMBMS [2], and most recently Advanced Television Systems Committee (ATSC) have worked on enabling DASH delivery via broadcast. Although previously File Delivery

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

Copyright is held by the owner/author(s).

MMSys'16, May 10-13, 2016, Klagenfurt, Austria

ACM 978-1-4503-4297-1/16/05.

<http://dx.doi.org/10.1145/2910017.2910640>

over Unidirectional Transport (FLUTE) protocol has been utilized to enable DASH broadcast delivery [2], this protocol was not designed for real-time object delivery required for broadcast purposes. To this end, ATSC is employing a new protocol named Real-time Object delivery over Unidirectional Transport (ROUTE) [1] which overcomes several hurdles faced by FLUTE for real-time object delivery.

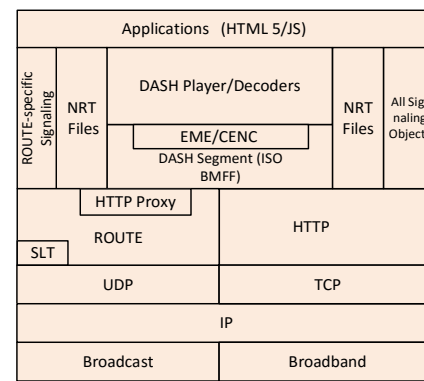


Figure 1: ATSC 3.0 “Conceptual protocol stack” [1] for ROUTE-DASH delivery

Figure 1 shows the relevant section of the conceptual protocol stack for an ATSC 3.0 receiver [1] where DASH delivery via ROUTE can enable

- Broadcast only,
- Hybrid broadcast/broadband, and
- Broadband only

service delivery. The detailed specification for realizing such services is provided in [1]. All these service configurations have been realized and demonstrated for ROUTE-DASH, as explained in the following sections.

2. DEMONSTRATOR ARCHITECTURE

2.1 Basic Sender and Receiver Setup

Figure 2 shows the basic sender and receiver configuration. On a high level, 2 channels are being sent, each with 1080p H.264/AVC video and HE AAC v2 audio. The data is sent over a local-area network via IP multicast, and a receiver can tune into one of the channels. The following sections provide more details about the sender and the receiver.

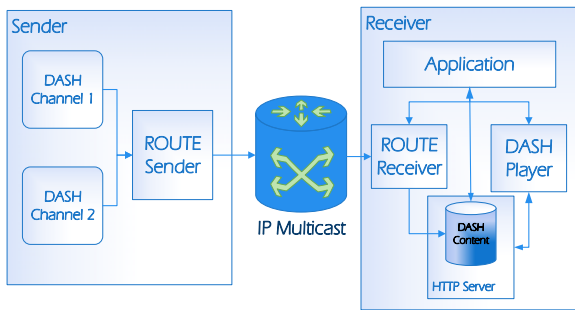


Figure 2: Basic sender and receiver configuration

2.2 Sender Setup

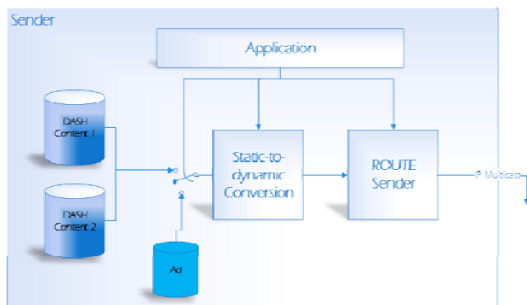


Figure 3: Sender architecture

Figure 3 shows the basic sender architecture. The design can input static DASH content in live profile as input. As shown here, two sets of DASH content, each representing part of an ATSC service can be fed into the system. Optionally, an advertisement content (also static, live profile) can be inserted into the main content at a configurable time.

Upon a user interaction with the application to start the service, a static to dynamic conversion module processes the static content:

1. The MPD is converted to dynamic according to the current time of start of sending.
2. Ad content reference, if requested, is inserted in the main MPD at its configured time.
3. Timing files are generated for ROUTE sender module that sends DASH Media Segments according to this timing information.

The ROUTE sender then pushes the content via IP multicast according to the service configuration information provided to the sender-side application.

2.3 Receiver Setup

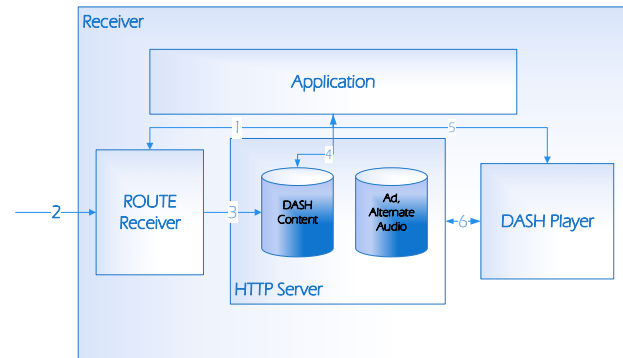


Figure 4: Basic receiver architecture

Figure 4 shows the basic receiver architecture. The interfaces are numbered according to their role as described with the corresponding numbering, in the following. In the following context, a tune-in is defined as the time when MPD, initialization information and at least one DASH Media Segment for each adaptation set is already received.

As the user selects the channel to play via the application,

1. The application configures the ROUTE receiver with IP addresses, ports, Transport Session Identifiers (TSI).
2. ROUTE receiver starts reception of the multicast data.
3. ROUTE places the delivery objects in a local HTTP server, it could be assisted via the application to complete this action.
4. The application monitors the reception of data:
 - i. Waits for the MPD reception.
 - ii. Based on MPD information, monitors when a “tune-in” has happened
 - iii. Based on tune-in time, process MPD (further details in Section 2.5)
 - a. Update DASH MPD@availabilityStartTime (AST) based on tune-in time, and the duration of the media segments.
 - b. Update Period@start, SegmentTemplate@startNumer, SegmentTemplate@personationTimeOffset
5. Start playback/play control (pause etc.)

Channel change:

If the user interacts with the application to change the channel, the following actions happen:

- Application pauses the DASH player via 5,
- stops ROUTE reception via 1,
- clear DASH content via 4,
- Starts from step 1.

For the receiver setup to work properly, the following must hold:

- i. Client, server have to be in time sync.
- ii. All processing at the receiver, including in the application has to be sub-second accurate.

2.4 MPD-Less Startup

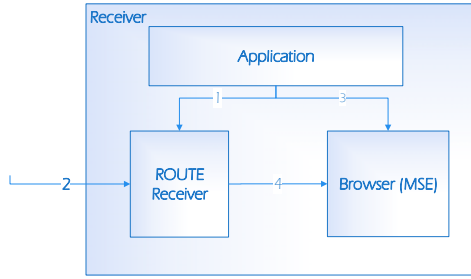


Figure 5: Receiver setup for MPD-less startup

ATSC specification [1] allow for sufficient information signaling that the tune-in process at the receiver can be completed without waiting for the reception of the MPD itself. This is also enabled by ROUTE specification by the specified Media Delivery Event (MDE) mode.

For this case, the ROUTE receiver is directly interfaced with the media decoders to push the media data to the decoders as soon as it is received. This allows for optimization in tune-in time.

2.5 MPD PROCESSING

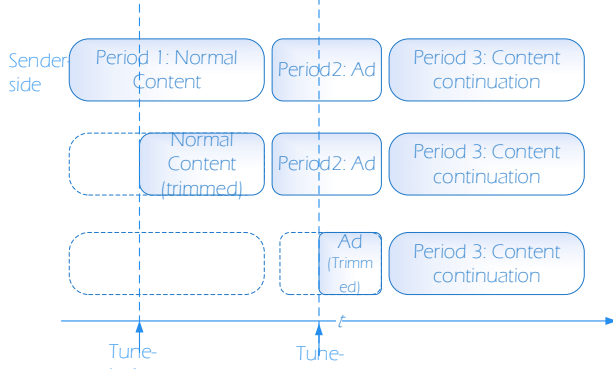


Figure 6: MPD at the sending end (top) and its processing for various tune-in times

As the receiver tunes into the content as described in Section 2.3, a conforming DASH player cannot play the content as it is, because the MPD is representing a data with segments before the tune-in time actually missing at the receiver. Hence, as described in Section 2.3, a Period is

- Left intact if its Period@start is after the tune-in time.
- Trimmed according to the tune-in time if the tune-in falls between the start of this Period and the next, as shown in Figure 6.
- Discarded if the tune-in is at or after the start of the next Period.

2.6 Ad-Insertion and Customization

As discussed in preceding sections, and advertisement can be inserted into the main content by using multiple periods. At the receiver side, this Ad can be replaced by another Ad if client is customized as such. This would mean replacing the Ad period in

the MPD with another period that points to a broadband address, where the Ad data is then fetched from an Ad server.

2.7 Hybrid Delivery

As described in Section 1, DASH delivery via ROUTE can enable hybrid services. In this case, some components in the MPD may point to broadcast delivery, while others (e.g. an alternate language audio track, a different video view, subtitles, etc.) may point to broadband resource. Such signaling may be embedded in the content from the sending side, but if allowed by the content provider, such compositions can also be done on the receiver side by updating the MPD as required.

Hybrid delivery can also be used to recover segment lost over the broadcast channel. Since DASH is loss-intolerant, such a lost segment must be recovered to enable service continuity. This recover can be done via broadband connection. For a connected receiver, for example, multiple base URLs can be signaled to the receiver. One of them is pointing to the broadcast, and the other to the broadband location. In this case, if the client received a HTTP 404 error for one or more segments of the broadcast content, implying that the segment is lost, it can attempt to recover this segment from broadband.

2.8 System Setup and User Interface

The setup for demonstration is shown in Figure 7. The sender is a Linux based machine, multicasting the sending data via multicast capable router. The receivers demonstrated are based on Android or PC (Windows/Linux). The whole setup can also be run on a single device if required.

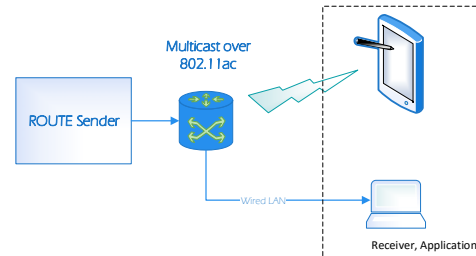


Figure 7: Overview of the system for demonstration

The source code for the complete setup is made available [5] via open-source licensing.

DASH source content is based on Creative Commons license (e.g. Tears of Steel, Sintel, etc.) DASH encoding is done by using FFmpeg [6] and MP4Box [7]. Some of the main parameters of the content are tabulated in Table 1.

Table 1: Content parameters

Type	Parameter	Value
Content	Duration	12 min 14 sec
DASH	Segment Sizes	500 msec, 1 second
Video	Codec	H.264/AVC
	Resolution	1920x1080p
	Frame rate	24 fps

	Bitrate	8 Mbps
Audio	Codec	AAC LC
	Channels	2
	Bitrate	64kbps

The application and static to dynamic conversion tool is implemented using web-based languages (Python, PHP, JavaScript). ROUTE sender and receiver is implemented in C++, and is based on the MAD FLUTE open-source project [8].

On the receiver side, an application based on JavaScript and PHP runs the setup. Apache HTTP server [10] hosts the local content, which can be played by DASH player [9], [11] via MSE-supporting browsers, such as Google Chrome browser.

2.8.1 Running the Demo

On the sender-side user interface the demonstration is run by

- Selecting the desired network interface.
- Selecting the desired Ad-insertion time.

And then turning the service slider to “on” position.

On the receiver demo page, turning the Unicast slider to “on” position enables the hybrid service, by enabling playback of multi-component audio, as shown in Figure 8.

In the player controls, selecting the gear icon will show the available audio options as shown in Figure 8. By default, using broadcast only (Unicast/Broadband disabled), both channels will show a single audio track (English). When the Unicast slider is turned to “on”, a second Italian audio track will appear in options for Tears of Steel (Channel 1). Selecting it will change the audio language. The default audio language selected after a tune-in is English. A synchronous audio-video playback is demonstrated.

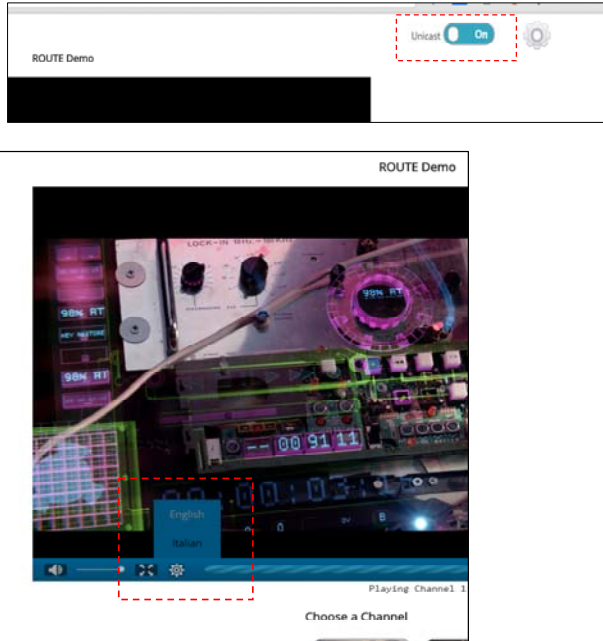


Figure 8: Receiver User Interface

3. PERFORMANCE

Based on the existing setup, using off-the-shelf, mid-range hardware, the performance is assessed based on Tune-in delay

$$\text{Tune-in delay} = \text{Tune-in time} - \bar{\Delta}$$

Where Tune-in time is measured for 32 tune-in attempts initiated at pseudorandomly selected time instances. $\bar{\Delta}$ is measured as the mean segment reception delay. Figure 9 shows the performance results.

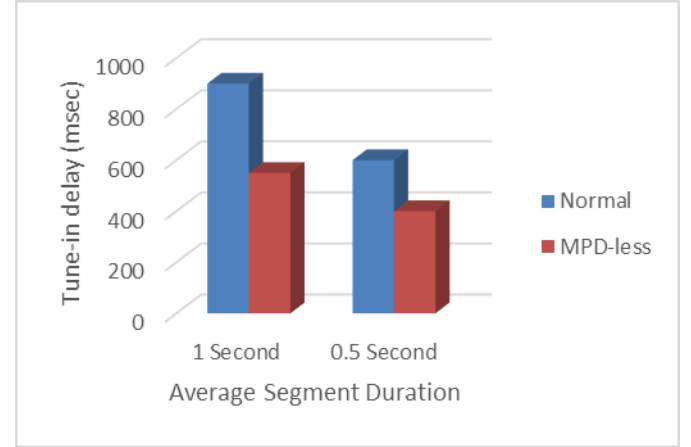


Figure 9: Performance results based on tune-in delay

The delays combine all theoretical and system-specific issues. The latter come from the operating system related delay issues, and hence it can be seen that halving the segment duration to 500 msec gives only ~30% performance improvement. MPD-less startup can provide an additional performance improvement of up to 40%.

4. Acknowledgement

Development work for this demonstration was funded by Qualcomm Inc.

5. REFERENCES

- [1] A/331, ATSC Candidate Standard: “Signaling, Delivery, Synchronization, and Error Protection,” 1.2016.
- [2] 3GPP TS 23.246 “Multimedia Broadcast/Multicast Service (MBMS); Architecture and functional description” 12.2015.
- [3] ETSI TS 103 285 V1.1.1 “Digital Video Broadcasting (DVB); MPEG-DASH Profile for Transport of ISO BMFF Based DVB Services over IP Based Networks,” 5.2015.
- [4] ISO/IEC 23009-1 “Information technology — Dynamic adaptive streaming over HTTP (DASH) Part 1: Media presentation description and segment formats,” 2nd Ed, 2014.
- [5] ATSC ROUTE DASH Demonstrator [online]: https://github.com/waqarz/ATSC_ROUTE
- [6] FFmpeg [online]: <https://www.ffmpeg.org/>
- [7] MP4Box [online]: <https://gpac.wp.mines-telecom.fr/mp4box/>
- [8] MAD Project's Home Page [online]: mad.cs.tut.fi/
- [9] dash.js [online]: <https://github.com/Dash-Industry-Forum/dash.js/>
- [10] Apache HTTP Server [online]: <https://httpd.apache.org>
- [11] Bitdash [online]: www.dash-player.com/