



Multimedia

Contents

2	Requirements	2
3	Hardware-accelerated media rendering	2
4	Multimedia Framework	3
5	Progressive download and buffered playback	3
6	Distributed playback support	3
7	Camera display on boot	4
8	Video playback on boot	4
9	Camera widget	4
10	Transcoding	4
11	DVD playback	4
12	Traffic control	5
13	Solutions	5
14	Multimedia Framework	5
15	Hardware-accelerated Media Rendering	6
16	Buffering playback in GStreamer and clutter-gst	6
17	Distributed playback	7
18	Camera and Video display on boot	7
19	Camera widget and clutter-gst	9
20	Transcoding	9
21	DVD playback	9
22	Traffic control	10

This document covers the various requirements for multimedia handling in the Apertis platform.

The FreeScale I.MX/6 platform provides several IP blocks offering low-power and hardware-accelerated features:

- GPU : For display and 3D transformation/processing
- VPU : For decoding and encoding video streams

The Apertis platform will provide robust and novel end-user features by getting the most out of those hardware components. However, in order to retain power efficiency, care must be taken in the way those components are exposed to applications running on the platform.

The proposed solutions outlined in this document have also been chosen for the Apertis platform to re-use as many “upstream”open-source solutions as possible, to minimize the maintenance costs for future projects based upon Apertis.

Requirements

Hardware-accelerated media rendering

The Apertis system will need to make usage of the underlying GPU/VPU hardware acceleration in various situations, mainly:

- 40 • Zero copy of data between the VPU decoding system and the GPU display
41 system
- 42 • Be usable in WebKit and with the Clutter toolkit
- 43 • Integration with FreeScale and ADIT technologies

44 **Multimedia Framework**

45 In a system like Apertis, writing a wide array of applications and end-user
46 features offering multimedia capabilities requires a framework which will offer
47 the following features:

- 48 • Handle a wide variety of use-cases (playback, recording, communication,
49 network capabilities)
- 50 • Support multiple audio, video and container formats
- 51 • Capability to add new features without having to modify existing applica-
52 tions
- 53 • Capability to handle hardware features with as little overhead as possible
- 54 • Widely adopted by a variety of libraries, applications and systems

55 In addition, this system needs to be able to handle the requirements specified
56 in **Hardware accelerated media rendering**.

57 **Progressive download and buffered playback**

58 The various network streams played back by the selected technology will need
59 to provide buffering support based on the playback speed and the available
60 bandwidth.

61 If possible a progressive download strategy should be used, using such a strategy
62 the network media file is temporarily stored locally and playback starts when it
63 is expected the media can be played back without a need to pause for further
64 buffering. Or in other words, playback starts when the remaining time to finish
65 the download is less then the playback time of the media.

66 For live media where progressive downloading is not possible (e.g. internet
67 radio) a limited amount of buffering should be provided to offset the effect of
68 temporary jitter in the available bandwidth.

69 Apart from the various buffering strategies, the usage of adapative bitrate
70 streaming technologies such as HLS or MPEG-DASH is recommended if avail-
71 able to continuously adapt playback to the current network conditions.

72 **Distributed playback support**

73 The Apertis platform wishes to be able to share playback between multiple
74 endpoints. Any endpoint would be able to watch the same media that another

75 is watching with perfect synchronization.

76 **Camera display on boot**

77 Apertis requires the capability to show camera output during boot, for example
78 to have rear camera view for parking quickly. Ideally, the implementation of
79 this feature must not affect the total boot time of the system.

80 **Video playback on boot**

81 Apertis requires the capability to show a video playback during boot. This
82 shares some points with the section **Camera display on boot** regarding the re-
83 quirements, the implementation, and risks and concerns. Collabora has some
84 freedom here to restrict the fps, codecs, resolutions, quality of the video to be
85 playback in order to be able to match the requirements.

86 **Camera widget**

87 Apertis requires that a camera widget that can be embedded to applications to
88 easily display/manipulate camera streams is provided. The widget should offer
89 the following features:

- 90 • Retrieve the list of supported camera devices and ability to change the
91 active device
- 92 • Support retrieving and updating color balance (saturation, hue, brightness,
93 contrast), gamma correction and device capture resolution
- 94 • Provides an interface for image processing
- 95 • Record videos and take pictures

96 **Transcoding**

97 *Transcoding* can be loosely described as decoding, optionally processing and re-
98 encoding of media data (video, audio, ...) possibly from one container format to
99 another. As a requirement for Apertis, transcoding must be supported by the
100 Multimedia Framework.

101 **DVD playback**

102 Most DVDs are encrypted using a system called **CSS**¹ (content scrambling sys-
103 tem), that is designed to prevent unauthorized machines from playing DVDs.
104 CSS is licensed by the DVD Copy Control Association (DVD CCA), and a CSS
105 license is required to use the technology, including distributing CSS enabled
106 DVD products.

¹<http://www.dvdcqa.org/css>

107 Apertis wishes to have a legal solution for DVD playback available on the plat-
108 form.

109 **Traffic control**

110 Traffic control is a technique to control network traffic in order to optimize or
111 guarantee performance, low-latency, and/or bandwidth. This includes deciding
112 which packets to accept at what rate in an input interface and determining
113 which packets to transmit in what order at what rate on an output interface.

114 By default traffic control on Linux consists of a single queue which collects
115 entering packets and dequeues them as quickly as the underlying device can
116 accept them.

117 In order to ensure that multimedia applications have enough bandwidth for
118 media streaming playback without interruption when possible, Apertis requires
119 that a mechanism for traffic control is available on the platform.

120 **Solutions**

121 **Multimedia Framework**

122 Based on the requirements, **we propose selection of the GStreamer mul-**
123 **timedia framework**², a LGPL-licensed framework covering all of the required
124 features.

125 The GStreamer framework, created in 1999, is now the de-facto multimedia
126 framework on GNU/Linux systems. Cross-platform, it is the multimedia back-
127 bone for a wide variety of use-cases and platforms, ranging from voice-over-
128 IP communication on low-power handsets to transcoding/broadcasting server
129 farms.

130 Its modularity, through the usage of plugins, allows integrators to re-use all the
131 existing features (like parsers, container format handling, network protocols,
132 and more) and re-use their own IP (whether software or hardware based).

133 Finally, the existing eco-system of application and libraries supporting
134 GStreamer allows Apertis to benefit from those where needed, and benefit
135 from their on-going improvements. This includes the WebKit browser, and the
136 Clutter toolkit.

137 **The new GStreamer 1.0 series will be used** for Apertis. In its 6 years
138 of existence, the previous 0.10 series exhibited certain performance bottlenecks
139 that could not be solved cleanly due to the impossibility of breaking API/ABI
140 compatibility. The 1.0 series takes advantage of the opportunity to fix the
141 bottlenecks through API/ABI breaks, so Apertis will be in a great position to
142 have a clean start.

²<http://gstreamer.freedesktop.org/>

143 Amongst the new features the 1.0 series brings, the most important one is related
144 to how memory is handled between the various plugins. This is vital to support
145 the most efficient processing paths between plugins, including first-class support
146 for zero-copy data passing between hardware decoders and display systems.
147 Several³ presentations⁴ are available detailing in depth the changes in the
148 GStreamer 1.0 series.

149 Hardware-accelerated Media Rendering

150 The current set of GStreamer plugins as delivered by Freescale targets the
151 Gstreamer 0.10 series, for usage with GStreamer 1.0 these plugins will need
152 to be updated.

153 As freescale was not able to deliver an updated set of plugins in a reasonable
154 timeframe Collabora has done a initial proof of concept port of the VPU plugins
155 to Gstreamer 1.0 allowing ongoing development of the middleware stack to focus
156 purely on Gstreamer 1.0.

157 Eventually it is expected that freescale will deliver an updated set of VPU
158 plugins for usage with Gstreamer 1.0.

159 to benefit as much as possible from improvements provided by the “upstream”
160 GStreamer in the future, it is recommend need to ensure that the platform-
161 specific development is limited to features specific to that platform.

162 Therefore it is recommended for the updated VPU plugins to be based on exist-
163 ing base video decoding/encoding classes (See [GstBaseVideoDecoder](#)⁵, [GstBaseVideoEncoder](#)⁶). This will ensure that:

- 165 • The update plugins will benefit from any improvements done in those base
166 classes and future adjustments to ensure proper communication between
167 decoder/encoder elements and other elements (like display and capture
168 elements).
- 169 • The updated plugins will benefit from commonly expected behaviors of
170 decoders and encoders in a wide variety of use-cases (and not just local file
171 playback) like QoS (Quality of Service), low-latency and proper memory
172 management.

173 Buffering playback in GStreamer and clutter-gst

174 [ClutterGstPlayer](#)⁷ uses the [playbin2](#)⁸ GStreamer element for multimedia content

³https://events.static.linuxfound.org/images/stories/pdf/lf_elc12_hervey.pdf

⁴<http://gstconf.ubicast.tv/videos/keynote-gstreamer10/>

⁵<https://www.freedesktop.org/software/gstreamer-sdk/data/docs/latest/gst-plugins-bad-libs-0.10/gst-plugins-bad-libs-GstBaseVideoDecoder.html>

⁶<https://www.freedesktop.org/software/gstreamer-sdk/data/docs/latest/gst-plugins-bad-libs-0.10/gst-plugins-bad-libs-GstBaseVideoEncoder.html>

⁷<http://developer.gnome.org/clutter-gst/stable/ClutterGstPlayer.html>

⁸<https://www.freedesktop.org/software/gstreamer-sdk/data/docs/2012.5/gst-plugins->

175 playback, which uses [queue2](#)⁹ element to provide the necessary buffering for both
176 live and on demand content. For the Apertis release (12Q4) new API was added
177 to clutter-gst to make it more easier for applications to correctly control this
178 buffer. Work is currently in progress to upstream these changes.

179 **Progressive buffering based on expected bandwidth** Depending on the
180 locality it might be desirable to not only buffer based on the currently available
181 bandwidth, but also on the expected bandwidth. For example the navigation
182 system may be aware of a tunnel coming up, where no or only very limited
183 bandwidth is available.

184 Due to the way buffering works in Gstreamer the final control for when playback
185 starts rests with the application, normally an application uses the estimates for
186 remaining download time provided by gstreamer (which is based on the current
187 download speed). In the case where the application has the ability to make a
188 more educated estimate by using location/navigation information, it can safely
189 ignore Gstreamers estimate and purely base playback start on its own estimate.

190 **Distributed playback**

191 As the basis for the distributed playback proof of concept solution Collabora
192 suggest the usage of the [Aurena](#)¹⁰ client/daemon infrastructure. Aurena is a
193 small daemon which announces itself on the network using avahi. This daemon
194 provides the media and control information over http and also provide provides
195 a Gstreamer based network clock for to use for clients to synchronize against.

196 Aurena will be integrated in the Apertis distribution an example clutter-gst
197 client will be provided.

198 As Aurena is an active project and further work on this topic is scheduled for the
199 Q2 of 2014, more details will be provided on the current state and functionality
200 available in Aurena closer to that time.

201 **Camera and Video display on boot**

202 In order to keep the implementation both low in complexity and flexible a pure
203 user-space solution is recommended, that is to say no kernel modification or
204 bootloader modification are done to enable this functionality.

205 The advantage of such a solution is that a lot of common userspace function-
206 ality can be re-used by the implemention. The main disadvantage is that this
207 functionality will only be available when userspace is started.

base-plugins-0.10/gst-plugins-base-plugins-playbin2.html

⁹<http://gstreamer.freedesktop.org/data/doc/gstreamer-plugins/html/gstreamer-plugins-queue2.html>

¹⁰<https://github.com/thaytan/aurena>

208 To provide a general feeling for the timings involved when running an unopti-
209 mized darjeeling image (130312) on the I.MX6 Sabrelite board the boot break-
210 down is as follows (Note that darjeeling isn't optimized for startup time) :

- 211 • 0.00s: Power plugged in
- 212 • 0.26s: u-boot started
- 213 • 1.23s: Kernel starting
- 214 • 4.12s: LVDS screen turns on
- 215 • 4.59s: Initramfs/mini userspace starting
- 216 • ~6.00s: Normal userspace starting.

217 The u-boot boot delay was disable for this test, no other changes

218 Even though these number should be improved by the boot optimisation work
219 (planned for Q2, 2013), the same order of magnitude will most likely remain for
220 the SabreLite hardware booting from MMC.

221 As a basis building block for providing this functionality [Plymouth](#)¹¹ will be
222 used. Plymouth is the de-factor application used for showing graphical boot
223 animations while the system is booting, being using by Fedora, Ubuntu and
224 many others. On most systems Plymouth takes advantage of the modesetting
225 DRM drivers, with fallbacks to using the old-style dumb framebuffer or even a
226 pure text mode.

227 Plymouth has a extensive pluggable theming system. New themes can
228 be written either in C or using a simple scripting language. A good
229 overview/introduction of the plymouth specific theme scripting can be found
230 in a series of [blog posts by Charley Brey](#)¹².

231 Plymouth has the ability to use themes which consists of a series of full-screen
232 images or in principle even a video file, however most boot animations are kept
233 relatively simple and are rendered on the fly using plymouths built-in image
234 manipulation support. The reason for this is simply an efficiency trade-of, while
235 on-the-fly rendering adds some cpu load for simpler animations that cpu load will
236 be still lower then loading every frame from an image file or rendering a video.
237 Furthermore this approach reduces the size and number of assets which have to
238 be loaded from storage. As such, to minimize the impact on boot performance
239 the use simple themes which are rendered on the fly is recommended over the
240 use of full-screen images or videos.

241 To add support for the “camera on boot” functionality plymouth will be extended
242 such that it can be requested to switch to a live-feed of the (rear-view) camera
243 during boot-up. To be able to support a wide range of cameras (e.g. both

¹¹<http://www.freedesktop.org/wiki/Software/Plymouth>

¹²<http://brej.org/blog/?p=158>

244 directly attached cameras and e.g. ip cameras) the use of Gstreamer is recom-
245 mended for this functionality. However to ensure boot speed isn't negatively
246 impacted Gstreamer can't be used from the initramfs as this would signifi-
247 cantly increase its size and thus slowing down the boot. An alternative to
248 using Gstreamer would be to implement dedicated, hardware/camera specific
249 plugins which are small enough to be included in the initramfs.

250 During Q2 of 2013 work will be done to optimise the boot time of Apertis. At
251 which point it will become more clear what the real impact of delaying camera-
252 on-boot until the start of full userspace is.

253 **Camera widget and clutter-gst**

254 To provide the camera widget functionality a new actor was developed for
255 clutter-gst. As any other clutter actor, the ClutterGstCameraActor can be em-
256 bedded in any clutter application and supports all requirements either through
257 the usage of provided convenience APIs or using GStreamer APIs directly. Im-
258 age processing is achieved with the usage of pluggable GStreamer elements.

259 **Transcoding**

260 GStreamer already supports [transcoding](#)¹³ of various different media formats
261 through the usage of custom pipelines specific to each input/output format.

262 In order to simplify the transcoding process and avoid having to deal with several
263 different pipelines for each supported media format, Collabora proposes adding
264 a new transcodebin GStreamer element which would take care of handling the
265 whole process automatically. This new element would provide a stand-alone
266 everything-in-one abstraction for transcoding much similar to what the play-
267 bin2 element does for playback. Applications could then take advantage of this
268 element to easily implement transcoding support with minimal effort.

269 **DVD playback**

270 [Fluendo DVD Player](#)¹⁴ is a certified, commercial software designed to reproduce
271 DVDs on Linux/Unix and Windows platforms allowing legal DVD playback on
272 Linux using GStreamer. It supports a wide range of features including, but not
273 limited to, full DVD playback support, DVD menu and subtitles support.

274 Other open-source solutions are available, but none of them meets the legal
275 requirements and for that Collabora proposes the usage of Fluendo DVD Player
276 and to provide the integration of it on the platform.

¹³<http://gentrans.sourceforge.net/docs/head/manual/html/howto.html#sect-introduction>

¹⁴<https://fluendo.com/en/products/multimedia/oneplay-dvd-player/>

277 Traffic control

278 Traffic control and shaping comes in two forms, the control of packets being
279 received by the system (ingress) and the control of packets being sent out by the
280 system (egress). Shaping outgoing traffic is reasonably straight-forward, as the
281 system is in direct control of the traffic sent out through its interfaces. Shaping
282 incoming traffic is however much harder as the decision on which packets to
283 sent over the medium is controlled by the sending side and can't be directly
284 controlled by the system itself.

285 However for systems like Apertis control over incoming traffic is far more im-
286 portant then controlling outgoing traffic. A good example use-case is ensuring
287 glitch-free playback of a media stream (e.g. internet radio). In such a case,
288 essentially, a minimal amount of incoming bandwidth needs to be reserved for
289 the media stream.

290 For shaping (or rather influencing or policing) incoming traffic, the only practi-
291 cal approach is to put a fake bottleneck in place on the local system and rely on
292 TCP congestion control to adjust its rate to match the intended rate as enforced
293 by this bottleneck. With such a system it's possible to, for example, implement
294 a policy where traffic that is not important for the current media stream (back-
295 ground traffic) can be limited, leaving the remaining available bandwidth for
296 the more critical streams .

297 However, to complicate matters further, in mobile systems like Apertis which
298 are connected wirelessly to the internet and have a tendency to move around it'
299 s not possible to know the total amount of available bandwidth at any specific
300 time as it's constantly changing. Which means, a simple strategy of capping
301 background traffic at a static limit simply can't work.

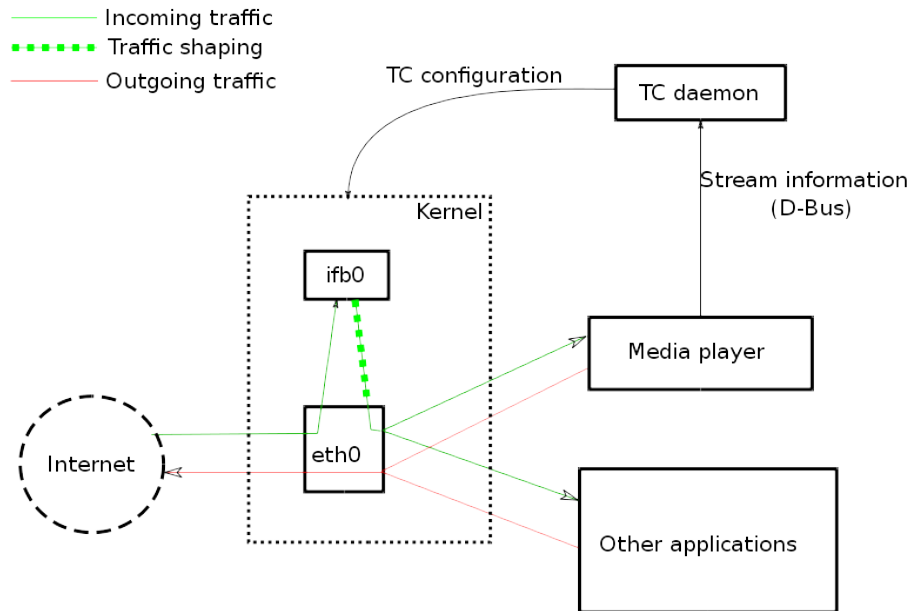
302 To cope with the dynamic nature a traffic control daemon will be implemented
303 which can dynamically update the kernel configuration to match the current
304 needs of the various applications and adapt to the current network conditions.
305 Furthermore to address the issues mentioned above, the implementation will
306 use the following strategy:

- 307 • Split the traffic streams into critical traffic and background traffic. Police
308 the incoming traffic by limiting the bandwidth available to background
309 traffic with the goal of leaving enough bandwidth available for critical
310 streams.
- 311 • Instead of having static configuration, let applications (e.g. a media player)
312 indicate when the current traffic rate is too low for their purposes. This
313 both means the daemon doesn't have to actively measure the traffic rate
314 and allows it cope with streams that don't have a constant bitrate more
315 naturally.
- 316 • Allow applications to indicate which stream is critical instead to properly
317 support applications using the network for different types of functionality

318 (e.g. a webbrowser). This rules out the usage of cgroups which only allows
 319 for per-process level granularity.

320 Communication between the traffic control daemon and the applications will be
 321 done via D-Bus. The D-Bus interface will allow applications to register critical
 322 streams by passing the standard 5-tuple (source ip and port, destination ip
 323 and port and protocol) which uniquely identify a stream and indicate when a
 324 particular stream bandwidth is too low.

325 To allow the daemon to effectively control the incoming traffic, a so-called In-
 326 termediate Functional Block device is used to provide a virtual network device
 327 to provide an artificial bottleneck. This is done by transparently redirecting the
 328 incoming traffic from the physical network device through the virtual network
 329 device and shape the traffic as it leaves the virtual device again. The reason for
 330 the traffic redirection is to allow the usage of the kernels egress traffic control to
 331 effectively be used on incoming traffic. The results in the example setup shown
 332 below (with eth0 being a physical interface and ifb0 the accompanying virtual
 333 interface).



334
 335 To demonstrate the functionality as describe above a simple demonstration me-
 336 dia application using Gstreamer will be written that communicates with the
 337 Traffic control daemon in the manner described. Furthermore some a testcase
 338 will be provided to emulate changing network conditions.