



Multimedia

1 Contents

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

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

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

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

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

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

37 Requirements

38 Hardware-accelerated media rendering

39 The Apertis system will need to make usage of the underlying GPU/VPU hard-
40 ware acceleration in various situations, mainly:

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

45 **Multimedia Framework**

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

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

56 In addition, this system needs to be able to handle the requirements specified
57 in [Hardware accelerated media rendering](#).

58 **Progressive download and buffered playback**

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

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

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

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

73 **Distributed playback support**

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

76 is watching with perfect synchronization.

77 **Camera display on boot**

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

81 **Video playback on boot**

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

87 **Camera widget**

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

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

97 **Transcoding**

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

102 **DVD playback**

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

¹<http://www.dvdcca.org/css.aspx>

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

110 **Traffic control**

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

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

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

121 **Solutions**

122 **Multimedia Framework**

123 Based on the requirements, **we propose selection of the GStreamer mul-**
124 **timedia framework²**, a LGPL-licensed framework covering all of the required
125 features.

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

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

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

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

²<http://gstreamer.freedesktop.org/>

144 Amongst the new features the 1.0 series brings, the most important one is related
145 to how memory is handled between the various plugins. This is vital to support
146 the most efficient processing paths between plugins, including first-class support
147 for zero-copy data passing between hardware decoders and display systems.
148 [Several](#)³ [presentations](#)⁴ are available detailing in depth the changes in the
149 GStreamer 1.0 series.

150 **Hardware-accelerated Media Rendering**

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

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

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

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

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

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

³<http://video.linux.com/videos/gstreamer-10-no-longer-compromise-flexibility-for-performance>

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

⁵<http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad-libs/html/gst-plugins-bad-libs-GstBaseVideoDecoder.html>

⁶<http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-bad-libs/html/gst-plugins-bad-libs-GstBaseVideoEncoder.html>

174 **Buffering playback in GStreamer and clutter-gst**

175 [ClutterGstPlayer](#)⁷ uses the [playbin2](#)⁸ GStreamer element for multimedia con-
176 tent playback, which uses [queue2](#)⁹ element to provide the necessary buffering
177 for both live and on demand content. For the Apertis release (12Q4) new API
178 was added to clutter-gst to make it more easier for applications to correctly
179 control this buffer. Work is currently in progress to upstream these changes.

180 **Progressive buffering based on expected bandwidth**

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

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

191 **Distributed playback**

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

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

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

202 **Camera and Video display on boot**

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

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

⁸<http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-base-plugins/html/gst-plugins-base-plugins-playbin2.html>

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

¹⁰<https://github.com/thaytan/aurena>

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

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

- 212 • 0.00s: Power plugged in
- 213 • 0.26s: u-boot started
- 214 • 1.23s: Kernel starting
- 215 • 4.12s: LVDS screen turns on
- 216 • 4.59s: Initramfs/mini userspace starting
- 217 • ~6.00s: Normal userspace starting.

218 The u-boot boot delay was disabled for this test, no other changes

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

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

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

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

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

¹²<http://brej.org/blog/?cat>

242 To add support for the “camera on boot” functionality plymouth will be ex-
243 tended such that it can be requested to switch to a live-feed of the (rear-view)
244 camera during boot-up. To be able to support a wide range of cameras (e.g.
245 both directly attached cameras and e.g. ip cameras) the use of Gstreamer is
246 recommended for this functionality. However to ensure boot speed isn’t neg-
247 atively impacted Gstreamer can’t be used from the initramfs as this would
248 significantly increase its size and thus slowing down the boot. An alternative to
249 using Gstreamer would be to implement dedicated, hardware/camera specific
250 plugins which are small enough to be included in the initramfs.

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

254 **Camera widget and clutter-gst**

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

260 **Transcoding**

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

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

270 **DVD playback**

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

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

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

¹⁴<http://www.fluendo.com/shop/product/fluendo-dvd-player/>

278 **Traffic control**

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

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

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

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

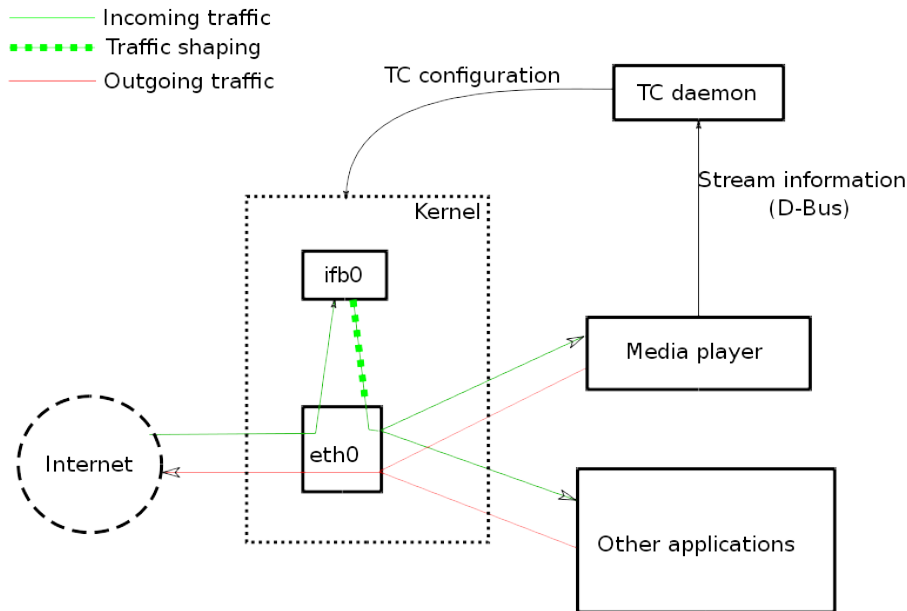
303 To cope with the dynamic nature a traffic control daemon will be implemented
304 which can dynamically update the kernel configuration to match the current
305 needs of the various applications and adapt to the current network conditions.
306 Furthermore to address the issues mentioned above, the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

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

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

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

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



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