Multimedia

Contents

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:

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

Multimedia Framework

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

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

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

Progressive download and buffered playback

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

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

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

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

Distributed playback support

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

is watching with perfect synchronization.

Camera display on boot

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

Video playback on boot

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

Camera widget

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

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

Transcoding

Transcoding can be loosely described as decoding, optionally processing and reencoding of media data (video, audio, …) possibly from one container format to another. As a requirement for Apertis, transcoding must be supported by the Multimedia Framework.

DVD playback

Most DVDs are encrypted using a system called CSS[1] (content scrambling system), that is designed to prevent unauthorized machines from playing DVDs. CSS is licensed by the DVD Copy Control Association (DVD CCA), and a CSS license is required to use the technology, including distributing CSS enabled DVD products.

[1] http://www.dvdcca.org/css

Apertis wishes to have a legal solution for DVD playback available on the platform.

Traffic control

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

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

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

Solutions

Multimedia Framework

Based on the requirements, **we propose selection of the GStreamer multimedia framework**[2], a LGPL-licensed framework covering all of the required features.

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

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

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

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

[2] http://gstreamer.freedesktop.org/

Amongst the new features the 1.0 series brings, the most important one is related to how memory is handled between the various plugins. This is vital to support the most efficient processing paths between plugins, including first-class support for zero-copy data passing between hardware decoders and display systems.

Several[3] presentations[4] are available detailing in depth the changes in the GStreamer 1.0 series.

Hardware-accelerated Media Rendering

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

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

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

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

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

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

Buffering playback in GStreamer and clutter-gst

ClutterGstPlayer[7] uses the playbin2[8] GStreamer element for multimedia con-

[3] https://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-

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

Progressive buffering based on expected bandwidth

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

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

Distributed playback

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

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

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

Camera and Video display on boot

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

The advantage of such a solution is that a lot of common userspace functionality can be re-used by the implemention. The main disavantage is that this 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/head/gstreamer-plugins/html/gstreamer-plugins-queue2.html

[10] https://github.com/thaytan/aurena

To provide a general feeling for the timings involved when running an unoptimized darjeeling image (130312) on the I.MX6 Sabrelite board the boot breakdown is as follows (Note that darjeeling isn't optimized for startup time) :

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

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

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

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

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

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

To add support for the "camera on boot" functionality plymouth will be extended such that it can be requested to switch to a live-feed of the (rear-view) camera during boot-up. To be able to support a wide range of cameras (e.g. both directly attached cameras and e.g. ip cameras) the use of Gstreamer is

[11] http://www.freedesktop.org/wiki/Software/Plymouth
[12] http://brej.org/blog/?p=158

recommended for this functionality. However to ensure boot speed isn't negatively impacted Gstreamer can't be used from the initramfs as this would significantly increase its size and thus slowing down the boot. An alternative to using Gstreamer would be to implement dedicated, hardware/camera specific plugins which are small enough to be included in the initramfs.

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

Camera widget and clutter-gst

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

Transcoding

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

In order to simplify the transcoding process and avoid having to deal with several different pipelines for each supported media format, Collabora proposes adding a new transcodebin GStreamer element which would take care of handling the whole process automatically. This new element would provide a stand-alone everything-in-one abstraction for transcoding much similar to what the playbin2 element does for playback. Applications could then take advantage of this element to easily implement transcoding support with minimal effort.

DVD playback

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

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

Traffic control

Traffic control and shaping comes in two forms, the control of packets being received by the system (ingress) and the control of packets being sent out by the

[13] http://gentrans.sourceforge.net/docs/head/manual/html/howto.html#sect-introduction
[14] https://fluendo.com/en/products/multimedia/oneplay-dvd-player/

system (egress). Shaping outgoing traffic is reasonably straight-forward, as the system is in direct control of the traffic sent out through its interfaces. Shaping incoming traffic is however much harder as the decision on which packets to sent over the medium is controlled by the sending side and can't be directly controlled by the system itself.

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

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

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

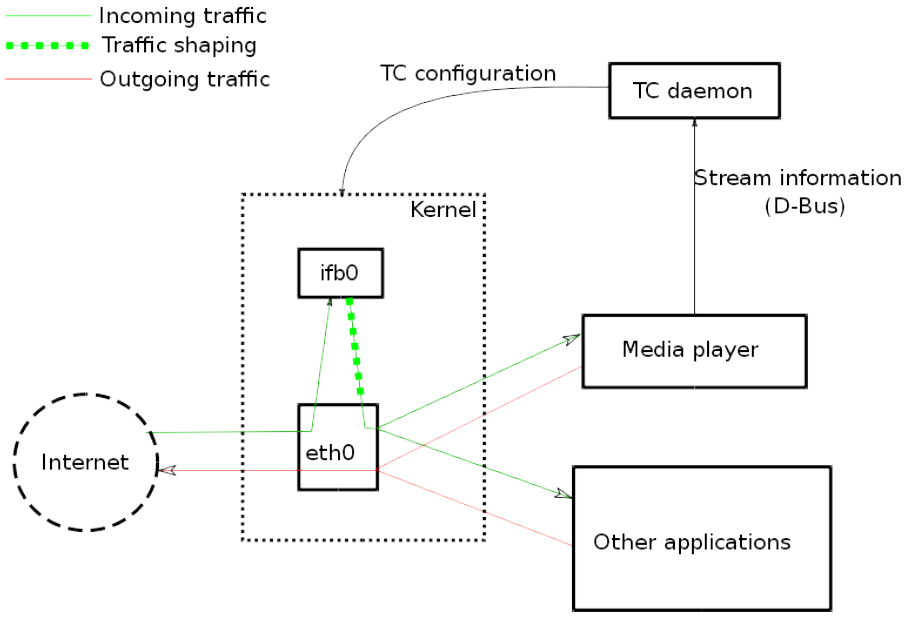
To cope with the dynamic nature a traffic control daemon will be implemented which can dynamically update the kernel configuration to match the current needs of the various applications and adapt to the current network conditions. Furthermore to address the issues mentioned above, the implementation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

- Split the traffic streams into critical traffic and background traffic. Police the incoming traffic by limiting the bandwidth available to background traffic with the goal of leaving enough bandwidth available for critical streams.
- Instead of having static configuration, let applications (e.g. a media player) indicate when the current traffic rate is too low for their purposes. This both means the daemon doesn't have to actively measure the traffic rate and allows it cope with streams that don't have a constant bitrate more naturally.
- Allow applications to indicate which stream is critical instead to properly support applications using the network for different types of functionality (e.g. a webbrowser). This rules out the usage of cgroups which only allows for per-process level granularity.

Communication between the traffic control daemon and the applications will be done via D-Bus. The D-Bus interface will allow applications to register critical

streams by passing the standard 5-tuple (source ip and port, destination ip and port and protocol) which uniquely identify a stream and indicate when a particular stream bandwidth is too low.

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



To demonstrate the functionality as describe above a simple demonstration media application using Gstreamer will be written that communicates with the Traffic control daemon in the manner described. Furthermore some a testcase will be provided to emulate changing network conditions.