## OPNFV DPACC Requirements

    WIKI is here: <https://wiki.opnfv.org/dpacc/dpacc_work_item_framework_arc>

Contents

[OPNFV DPACC Requirements 1](#_Toc437952434)

[1. Definitions 2](#_Toc437952435)

[3. Business requirements 3](#_Toc437952436)

[4. DPACC High Level Requirements 3](#_Toc437952437)

[5. Portability requirements 3](#_Toc437952438)

[6. Open Source requirements 5](#_Toc437952439)

[7. g-API: High Level Requirements 6](#_Toc437952440)

[8. legacy-API: High Level Requirements 7](#_Toc437952441)

[9. SIO: High Level Requirements 8](#_Toc437952442)

[10. HIO: High Level Requirements 8](#_Toc437952443)

[11. Best Known Methods and Guidelines 8](#_Toc437952444)

[12. Performance Metrics and Verification 9](#_Toc437952445)

[13. History 10](#_Toc437952446)

# Definitions

|  |  |
| --- | --- |
| Accelerator | Any software or hardware design able to accelerate packet processing over the current standard designs |
| Application Binary Interface (ABI) | An ABI (Application Binary Interface) is the set of runtime interfaces exposed by a library. It is similar to an API (Application Programming Interface) but is the result of compilation. It is also effectively cloned when applications link to dynamic libraries. That is to say when an application is compiled to link against dynamic libraries, it is assumed that the ABI remains constant between the time the application is compiled/linked, and the time that it runs. Therefore, in the case of dynamic linking, it is critical that an ABI is preserved, or (when modified), done in such a way that the application is unable to behave improperly or in an unexpected fashion.  Here is the link to DPDK ABI text: <http://dpdk.org/browse/dpdk/tree/doc/guides/contributing/versioning.rst> |

1. References

  WIKI is here: <https://wiki.opnfv.org/dpacc/dpacc_work_item_framework_arc>

# Business requirements

|  |  |
| --- | --- |
| BR-001 | DPACC MUST be targeted at driving open-source implementation of a generic acceleration framework for NFV acceleration |
| BR-002 | SHOULD ensure VNF vendors can leverage all accelerators exposed by the NFVI in a hardware independent way |
| BR-003 | SHOULD allow a NFVI operator to install a hardware or software accelerator not known at build time (Hot plug support) |

# DPACC High Level Requirements

|  |  |
| --- | --- |
| HLR-001 | (application portability) MUST provide portability for the applications |
| HLR-002 | SHOULD support binary portability of VNF VMs within the same instruction set architecture(Rob to Update) |
| HLR-002a | SHOULD provide a versioned ABI so that VNFs can leverage any acceleration under that ABI version without need to recompile and revalidate the VNF.  Note that VNFs including platform specific drivers should also be supported (with orchestration directing the VNF to a platform with the required features). There is an engineering trade-off between binary portability and performance from translating to/from an ABI standard. |
| HLR-003 | MUST have a API ABI requirement to track API/structure changes |
| HLR-004 | MUST preserve ABI if dynamically linked (when modified), done in such a way that the application is unable to behave improperly or in an unexpected fashion. |
| HLR-005 | (scalability) SHOULD support multiprocessing. E.g utilisation of multiple processor cores. |
| HLR-006 | (compatibility) MUST support legacy VNFs |
| HLR-006a | Must not impact the compatibility of legacy VNFs of the same architecture, (i.e. NFs running in the guest kenel/userspace. |

# Portability requirements

|  |  |
| --- | --- |
| POR-001 | MUST support Linux systems in Host and Guest |
| POR-001a | MUST follow the supported Linux OS platform in OPNFV releases |
| POR-001b | Should be OS agnostic  Editors Note: Contradiction to POR-001 |
| POR-001c | MUST follow the supported virtualization solutions in OPNFV releases |
| POR-001d | Should be virtualization-technique agnostic |
| POR-002 | SHOULD NOT expose the guest OS as part of the APIs  Note: Issues in implementations can be reported and scheduled for resolution |
| POR-003 | MUST be CPU Architecture agnostic  e.g ARM, IBM Power, MIPS and IA etc. |
| POR-004 | MUST be agnostic to CPU and system architectures  e.g CPU + External NIC or Integrated SoC etc. |
| POR-005 | MUST be I/O architecture agnostic  e.g MUST support PCI and non-PCI device configurations |
| POR-006 | MUST allow for different programming models,  e.g. Event model, run-to-completion, etc. |

# Open Source requirements

|  |  |
| --- | --- |
| OSR-001 | MUST be written in a portable language  e.g 'C' is the most common language and can be access from most other languages |
| OSR-002 | MUST upstream any changes to the Linux Kernel |
| OSR-003 | The DPACC implementation MUST be independent of non-upstreamed kernel modules or kernel modifications for the platform independent core DPACC implementation |
| OSR-003a | Any critical kernel modules must be upstreamed into Linux unless an optional module is used. All modules MUST be open sourced if required for the platform-independent core of the DPACC implementations.. |
| OSR-003b | For avoidance of doubt, platform-specific software and firmware including SoC/CPU/NIC firmware, drivers, etc., which plug into common API layers or glue code, need not be published in source code form to be used in conjunction with DPACC contributed API or glue code.  Editors Note: Rewrite to requirement or move to best practice section. |
| OSR-004 | MUST document DPACC API and open source code with Doxygen. |
| OSR-004a | Doxygen is the best known method for documenting open source code and MUST be used for all function headers, structures, structure member, macros and file headers to help in maintaining the code along with helping everyone to understand the code. |
| OSR-004b | Conditional OSR-004a Only applies to all open source code given to the DPACC project and not for private implementations. |
| OSR-004c | Documentation MUST clearly differentiate between public API and private definitions.  Editor: Added “Documentation” |

# g-API: High Level Requirements

Introduction

The following g-API is for the application portability and not specific to a specific software acceleration layer design.

|  |  |
| --- | --- |
| G-001 | **g-API MUST support multiple vendors and implementations at the same time to prevent vendor lock-in by hiding implementation details from the applications** |
| G-001a | Data exchanged via g-API MUST be generic and the underlying routines MAY need to convert it into a hardware specifihc format) |
| G-001b | g-API MAY provide abstract or non-abstract data types if required, the goal is to make the g-API usable by the VNF application only not to favor a specific s-API design.  Editors Note: Contradicting, clarify. |
| G-001c | g-API MAY provide accessor functions to simplify data access, but is not required to provide these types of APIs in favor of a specific s-API design |
| G-002 | MUST allow for deterministic execution and the best performance of the underlying Acceleration Core |
| G-002a | g-API **MUST** not introduce an undue overhead over native AC implementations, as measured with representative examples. The suggestion is less than 2% overhead. |
| G-002b | g-API **should not exclude**  underlying s-APIs from being accessible |
| G-003 | g-API SHOULD (only) expose API operations that are useful to the end application(s) and widely supported across different underlying hardware and software implementations.  Editors Note: If we are responsible for writing the spec then we indicate which operations are exposed = “useful”, thus this text should be removed. |
| G-003a | g-API should define the behaviour of API calls to be sufficiently generic and flexible to accommodate a reasonable range of hardware and software implementations |
| G-004 | gAPI SHOULD (only) expose API operations that are useful to the end application(s) and widely supported across different underlying hardware and software implementations. |
| G-004a | g-API should define the behaviour of API calls to be sufficiently generic and flexible to accommodate a reasonable range of hardware and software implementations |

|  |  |
| --- | --- |
| G-005 | gAPI SHOULD allow an application to query availability of a feature, where appropriate to support portability. |
| G-006 | g-API SHOULD define all possible errors cases are strictly defined and there’s no room for “unspecified” behavior unless performance is affected  Note: The intent is **NOT** to make a bulletproof API with extensive parameter checking, but to clearly define semantics of an API call   * + 1. e.g. in documentation, naming of API calls, doxygen etc.     2. This conflicts with the performance goals stated in numerous earlier points. You cannot have precisely specified portable error behavior with arbitrary ill-formed parameters unless the API implementations do extensive run-time parameter checking. This point needs to be clarified. Keith:I believe Ola added this statement, but not sure, I updated it to SHOULD and added the performance point.     3. 9.1 and 9.1.1 clarify that there is not a blanket requirement for run-time parameter checking since it may be necessary to make performance trade-offs.(Please add your name to the list with the correct color highlight, as I do not know who this is here. Thanks) |

# legacy-API: High Level Requirements

API is for legacy applications portability

|  |  |
| --- | --- |
| L-001 | MUST NOT require any changes or prevent usage of these APIs |
| L-001a | SHOULD support a reasonable set of API types sockets, libcrypto, ... |
| L-001b | MUST be documented if any differences from the native API |
| L-001c | Should clarify that legacy APIs may not exhibit the same performance characteristics as g-API usage. There is no "free lunch" here. It is expected that applications will, over time, migrate to use g-APIs to obtain best portability and performance. (OK, would this comment be OK to leave in the text then?) |

# SIO: High Level Requirements

|  |  |
| --- | --- |
| SIO-001 | MUST provide at least one guest to/from host network interface |
| SIO-002 | **MUST account for security concerns forchanges to VirtIO** |
| SIO-002a | **MUST** be backward compatible to older versions of VirtIO(need to pick a version) |
| SIO-002b | Need to address the backward compatibility in the case of Host upgrades and guest VNF not being upgraded this is the normal case. |

# HIO: High Level Requirements

|  |  |
| --- | --- |
| HIO-001 | MUST allow VirtIO as the fallback if passthru is not present |
| HIO-002 | SHOULD support PCI and non-PCI device pass throughs |
| HIO-002a | MAY? support features like SR-IOV and other pass-through designs |
| HIO-003 | MUST have discoverable devices via configuration or able to scan for devices, if pass-through is supported |
| HIO-004 | SHOULD support hot pluggable devices or non-direct hardware devices  Note: The goal for hotplug is to allow devices that support the feature to be hotplugged with software support. |
| HIO-005 | SHOULD NOT require the guest to support hotplug of devices |

Notes:  
    g-API: Need to define the application use cases to benchmark the application performance,

* e.g. L3 Forwarding using LPM, IP Fragmentation/Reassemble and in a virtual function application benchmark.

# 11. Best Known Methods and Guidelines

Duplicates here from the original requirements – to edit…

1. g-API MUST support multiple vendors and implementations at the same time to prevent vendor lock-in by hiding implementation details from the applications
2. Need to use best known methods for portability for code and APIs
3. **SHOULD NOT require modification to existing applications**
   1. For avoidance of doubt, the g-API may specify new APIs, to maximize portability, but developers are free to use legacy/s-APIs directly, with the understanding that application portability and functionality may be compromised to some degree.
4. **g-API SHOULD use software best-practice to decouple applications from software/hardware implementation specific data structures and implementation specific assumptions about the location of data**.
5. **g-API SHOULD provide explicit create/allocate and destroy/free for resources that are intrinsic to data-plane processing**
   1. e.g. for example: buffers and timers) to allow flexibility of implementation.Lingli: is this one belongs to the group of "handlers"? Shall we move it to the #5, rather than #4.6?
      1. All g-API data structures **SHOULD** be explicitly allocated and freed using the corresponding g-API allocators where required
6. In regards to APIs having general utility across applications and hardware spectrum (g-API #6)
   1. If a given functionality is not supported by the underlying design then the design should return NOT SUPPORTED as an error.
      1. Design guideline: Requiring a design to implement a functionality which can not be supported or is hardware supported is not reasonable and the design should be able to return not supported.
      2. Optional APIs and features simply promote fragmentation, which goes against the portability goals. gAPI should avoid optional features (at least for the first few releases).
         1. Note that this does not imply that every API will exhibit the same performance characteristics across every implementation. It should be assumed that there will be platform-specific variances in this area, however the goal should be that APIs should be efficiently implementable across all platforms.
7. **g-API must clearly define in documentation the behaviour of API calls**
   1. Including success/error cases with consideration of performance across multiple possible implementations
      1. e.g. Queue enqueue operation: “success” would indicate that the item was placed in the queue, but no guarantees that the receiver will ever process it (the receiver may crash and queue be destroyed before item is processed)

# Performance Metrics and Verification

<capture the performance metrics here>

# History

|  |  |
| --- | --- |
| 15th December 2015 | Initial draft from google docs version dated 14th December 2015. |
|  |  |

1. **Editors**

Keith Wiles

Bob Monkman

Michael Rooke

Bill Fischofer

1. **Contributors:**

Francois-Frederic Ozog

Josh Fender

Rob Dimond

Srinivasa Addepalli

Bob Monkman

Argy Krikelis

Bill Fischofer

Lingli Deng

Ola Liljedahl

Michael Rooke