## \*\*Editor: Keith Wiles;

**\*\*Contributors:**  
**Francois-Frederic Ozog, Josh Fender, Rob Dimond, Srinivasa Addepalli,**  
**BobMonkman, Argy Krikelis, Bill Fischofer, Lingli Deng, Ola Liljedahl, Michael Rooke.**  
  
    Font size is 14 and using Header 1 for titles.  
    WIKI is here: <https://wiki.opnfv.org/dpacc/dpacc_work_item_framework_arc>

## Please follow the guidelines in RFC 2119 : <https://www.ietf.org/rfc/rfc2119.txt>

Nokia Proposal – reordered requirements with headings.

# Definitions

# Business requirements

2. **to**

# DPACC High Level Requirements:

1. **(performance) MUST have a high performance design**
   1. Support deterministic performance for carrier grade network applications
2. **(application portability) MUST provide portability for the applications**
   1. MUST be source code compatible across various CPU architectures and binary compatible within a CPU architecture
      1. Source code portability is a **MUST** have goal
      2. Binary portability of vNF VMs across same instruction architectures is one of the goals
3. **MUST have a API ABI requirement to track API/structure changes**
4. **(scalability) MUST be scalabl~~e~~ in performance and design**
   1. e.g. Be able to demostrate some type of performance and scalability on number of cores and systems
5. **(compatibility) MUST support legacy VNFs**
   1. Must not impact the compatibility of legacy VNFs (i.e. NFs running in the guest kenerl/userspace.
6. **MUST NOT use non-upstreamed kernel modules or modifications for core DPACC system**
   1. Any critical kernel modules must be upstreamed into Linux unless an optional module is used. All modules MUST be open sourced if required for DPACC.
7. **(documentation) MUST document the API and code with Doxygen**

# Portability requirements

1. **MUST support Linux systems in Host and Guest[https://mail.google.com/mail/u/0/#inbox/1501457c6792146c](https://mail.google.com/mail/u/0/" \l "inbox/1501457c6792146c)** ???
   1. Support for red hat and ubuntu (Need create a list of supported Linux systems)
   2. List the supported DPACC systems to test/support
   3. Needs to support KVM/QEMU (List the oldest version supported)
   4. VMware, Hyper-V, Xen, ...
2. **SHOULD NOT expose the guest OS as part of the APIs**
   * 1. e.g. Wind River VxWorks, Microsoft Windows, Apple OS X, Cloudious OSv, BSD, Linux, ... to implement the DPACC architecture
     2. Without finding and verifying every version of every operating system we can not say must, but needs to be 'May'
3. **MUST support CPU Architectures**
   1. e.g. ARM, IBM Power, MIPS and IA
4. **MUST be agnostic to CPU and system architectures**
   1. Need to use best known methods for portability for code and APIs
5. **MUST support portability of applications across different CPU and system architecture**
   1. Enable multi-sourcing of infrastructure across multiple HW vendors who compete on performance, price and power (lowest TCO)
6. Bus Interface Requirements
   1. **MUST support PCI and non-PCI device configurations**
7. **MUST allow for different programming models, e.g. Event model, run-to-completion, ...**
   1. Must not restrict the application to a given programming model

# Open Source requirements

# Working Process Related

Comments marked design principle or working process should be moved to this section,

* 1. ~~DPACC must follow OPNFV IP Policy except for "downloadable device firmware"~~

# g-API: High Level Requirements

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

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**
   1. Data exchanged via g-API must be generic and standard (e.g. specified by RFC or other standard document, and the underlying routines may need to convert it into a hardware specific format)
   2. 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.
   3. 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
2. **SHOULD NOT require modification to existing applications**
   1. For avoidance of doubt, the g-API may specifiy new APIs, to maxmize 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.
3. **MUST allow for deterministic execution and the best performance of the underlying Acceleration Core**
   1. Which means a very thin layer on top of the s-API to not effect native Acceleration Core performance
      1. g-API does not require Real Time only deterministic exection or functionality
   2. g-API **MUST** include all underlining s-APIs to be accessable via the g-api as long as these are not vendor or s-API specific
   3. g-API **MUST** not introduce an undue overhead over native AC implementations, as measured with representative examples. The suggestion is less than 2% overhead.
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 allocate and freed using the corresponding g-API allocators where required
6. **gAPI SHOULD (only) expose API operations that are useful to the end application(s) and widely supported across different underlying hardware and software implementations.**
   1. gAPI **SHOULD** provide explicit create/allocate and destroy/free for resources that are intrinsic to data-plane processing (for example: buffers and timers) to allow flexibility of implementation.
      1. If a given functionality is not supported by the underlying design then the design should return NOT SUPPORTED as an error.
      2. 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.
      3. 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 SHOULD define the behaviour of API calls to be sufficiently generic and flexible to accomodate a reasonable range of hardware and software implementations**
   1. g-API SHOULD allow flexibility in definition so that hardware features can be used natively.
      1. A narrow description of functionality (that matches exactly the hardware of a single vendor) can cause lock-in as other vendors need to emulate or ‘fix up’ in software.
   2. gAPI SHOULD allow an application to query availability of a feature, where appropriate to support portability.
8. **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)
9. **g-API SHOULD define all possible errors cases are strictly defined and there’s no room for “unspecified” behavior unless performance is effected**
   1. 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 paramater 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

1. **MUST NOT require any changes or pervent usage of these APIs**
   1. SHOULD support a reasonable set of API types sockets, libcrypto, ...
   2. MUST be documented if any differences from the native API
   3. 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

1. **MUST provide at least one guest to/from host network interface**
2. **MUST upstream any changes to the Linux Kernel**
3. **MUST account for security concerns forchanges to VirtIO**
   1. **MUST** be backward compatible to older versions of VirtIO(need to pick a version)
   2. 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

1. **MUST allow VirtIO as the fallback if passthru is not present**
2. **SHOULD support PCI and non-PCI device pass throughs**
   1. **support features like SR-IOV and other pass-through designs**
3. **MUST have discoverable devices via configuration or able to scan for devices, if passthru is supported**
4. **SHOULD support hot pluggable devices or non-direct hardware devices**
   1. **The goal for hotplug is to allow devices that support the feature to be hotplugged with software support.**
5. **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. L3Forwarding using LPM, IP Fragmentation/Reassemble and in a virtual function application benchmark.

\*  
  
\*  
  
\*