Authors: Authors: Subhashini Venkataramanan, Srini Addepalli 6/2015

API Guidelines

Basic Types

Linux type definitions shall be followed. For example int, u64, char etc. shall be used.

API Naming convention

APIs are named such that they have the accelerator name, type and function. For instance, an ipsec look aside accelerator's SA creation function shall be named as g_ipsec_la_sa_add(). 'ipsec' refers to the accelerator name, 'la' indicates the type as look aside and sa_add is the actual function. At all times the object precedes the operation, as in this case 'sa' precedes 'add'.

Variable Naming convention

Naming convention for variables shall follow Linux style, readable and separated by underscore, when necessary.

Function Arguments and Return Values

All APIs return a value of SUCCESS or FAILURE.

For control or setup APIs that are used to setup states in the hardware accelerator it is preferable to use data structures to pass input and output parameters. While these setup or control functions do not come in the data path and hence do not impact performance, having parameters defined as structures enables extensibility in future without changing API prototypes. Structure introduced for passing in as parameters for functions shall have the function name as prefix and inargs/outargs as suffixes to indicate input and output arguments. For example, the input argument to g_ipsec_la_sa_add() would be `g_ipsec_la_sa_add_inargs and g_ipsec_la_sa_add_outargs.`

For data processing APIs, data structures are avoided in the packet processing calls and linear buffers are used with performance considerations in mind.

APIs shall also have flags to modify API behavior such as synchronous/asynchronous, response expected or not.

For example a set up API for setting up SAs would be as follows:

```
int g_ipsec_la_sa_add(
       struct g_ipsec_la_handle *handle, /* Accelerator handle */
         const struct g_ipsec_la_sa_add_inargs *in, /* Input */
         enum g_ipsec_la_control_flags flags, /* API flags */
         struct g_ipsec_la_sa_add_outargs *out /* Output */,
         struct g_ipsec_la_resp_args resp /* response callback in case
```

```
asynchronous mode with response flag is set */ );
```

In the above API, g_ipsec_la_control_flags and g_ipsec_la_resp_args are defined as follows:

```
enum g_ipsec_la_control_flags
{
      G_IPSEC_LA_CTRL_FLAG_ASYNC, /* If Set, API call be asynchronous.
Otherwise, API call will be synchronous */
      G_IPSEC_LA_CTRL_FLAG_NO_RESP_EXPECTED, /* If set, no response is
expected for this API call */
};

struct g_ipsec_la_resp_args
{
      struct g_ipsec_la_resp_cbfn cb_fn;
            /* Callback function if
               ASYNC flag is chosen */
      void *cb_arg;
      int32_t cb_arg_len; /* Callback argument length */
}
```

Application can request the response to be returned synchronously or asynchronously (G_IPSEC_LA_CTRL_FLAG_ASYNC). If the response is requested asynchronously, then the application should provide a callback function pointer and callback argument.

Also, in some scenarios, the API layer may have to do additional operations to force a response from the backend. The flag G_IPSEC_LA_CTRL_FLAG_NO_RESP_EXPECTED can be used by application to indicate whether the application should force the response from the backend or not .

A packet processing API in the case of IPSec would be as follows:

```
Prototype:
int32_t g_ipsec_la_packet_encap(
            struct g_ipsec_la_handle *handle,
            struct g_ipsec_la_control_flags flags,
            struct g_ipsec_la_sa_handle *handle; /* SA Handle */
            uint32 num_sg_elem; /* num of Scatter Gather elements */
            struct g_ipsec_la_data in_data[];
                  /* Array of data blocks */
            struct g_ipsec_la_data out_data[];
                  /* Array of output data blocks */
            struct g_api_resp_args resp)
```

In the above API, g_ipsec_la_data is defined as follows:

```
struct g_ipsec_la_data {
      uint8_t *buffer; /* Buffer pointer */
      uint32_t length; /* Buffer length */
}
```

API Types

APIs can be classified as management APIs and functional APIs.

Management APIs include APIs that VNF applications can use to find out about available accelerators, accelerator usage request and relinquish.

Functional APIs include Control or setup APIs for setting up state in the stateful hardware accelerator and and data processing APIs for packet processing.

Typically for any stateful hardware accelerator, the following APIs would be made available for control or setup of states

add – To add a state in the hardware accelerator

mod – To modify a state in the hardware accelerator

del – To delete a state in the hardware accelerator

get – get the current state as seen and maintained by the hardware accelerator; get types include get-first, get-next, get-exact etc.

Example: g-APIs for IPsec

G-APIs for IPsec are defined to allow VNF application access underlying hardware accelerator to perform IPSec accelerator operations.

G-APIs for IPSec shall include the following:

g_ipsec_la_open(), g_ipsec_la_close(), g_ipsec_la_sa_add(), g_ipsec_la_sa_del(), g_ipsec_la_sa_mod(), g_ipsec_la_sa_get(), g_ipsec_la_packet_encap(), g_ipsec_la_packet_decap(), g_ipsec_la_multi_packet_encap(), g_ipsec_la_multi_packet_decap().