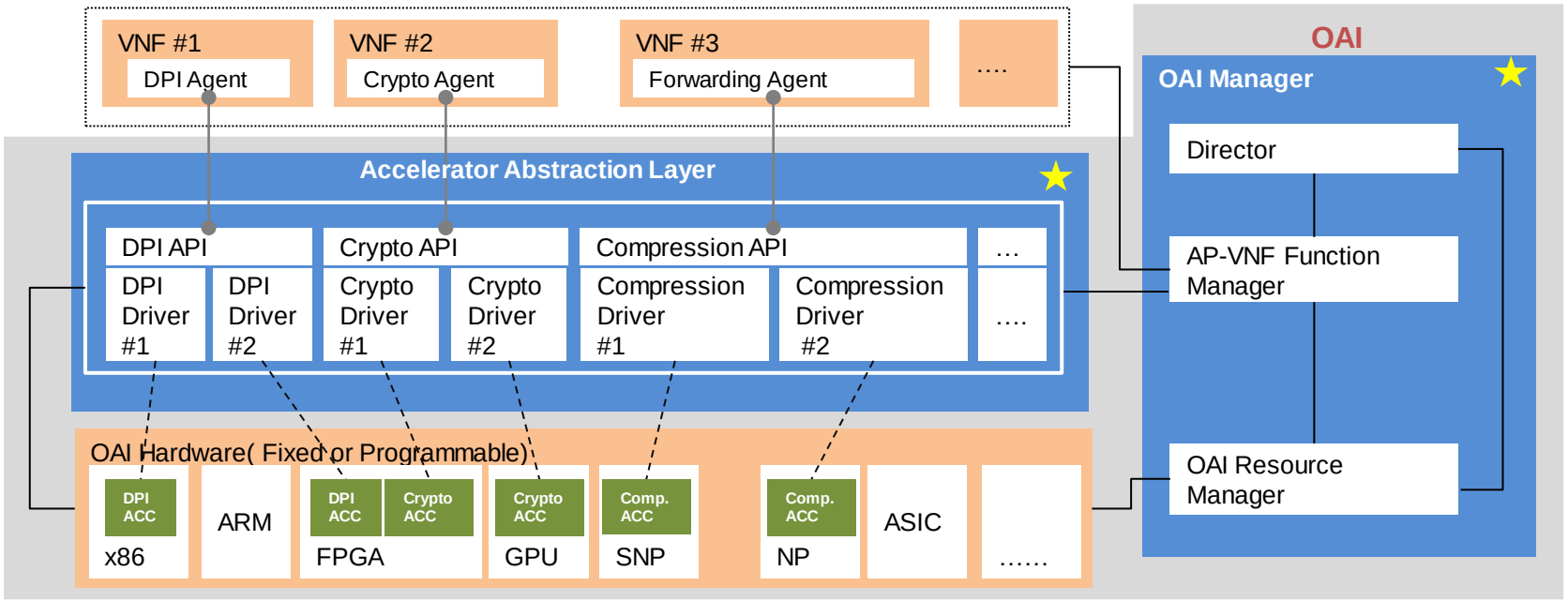
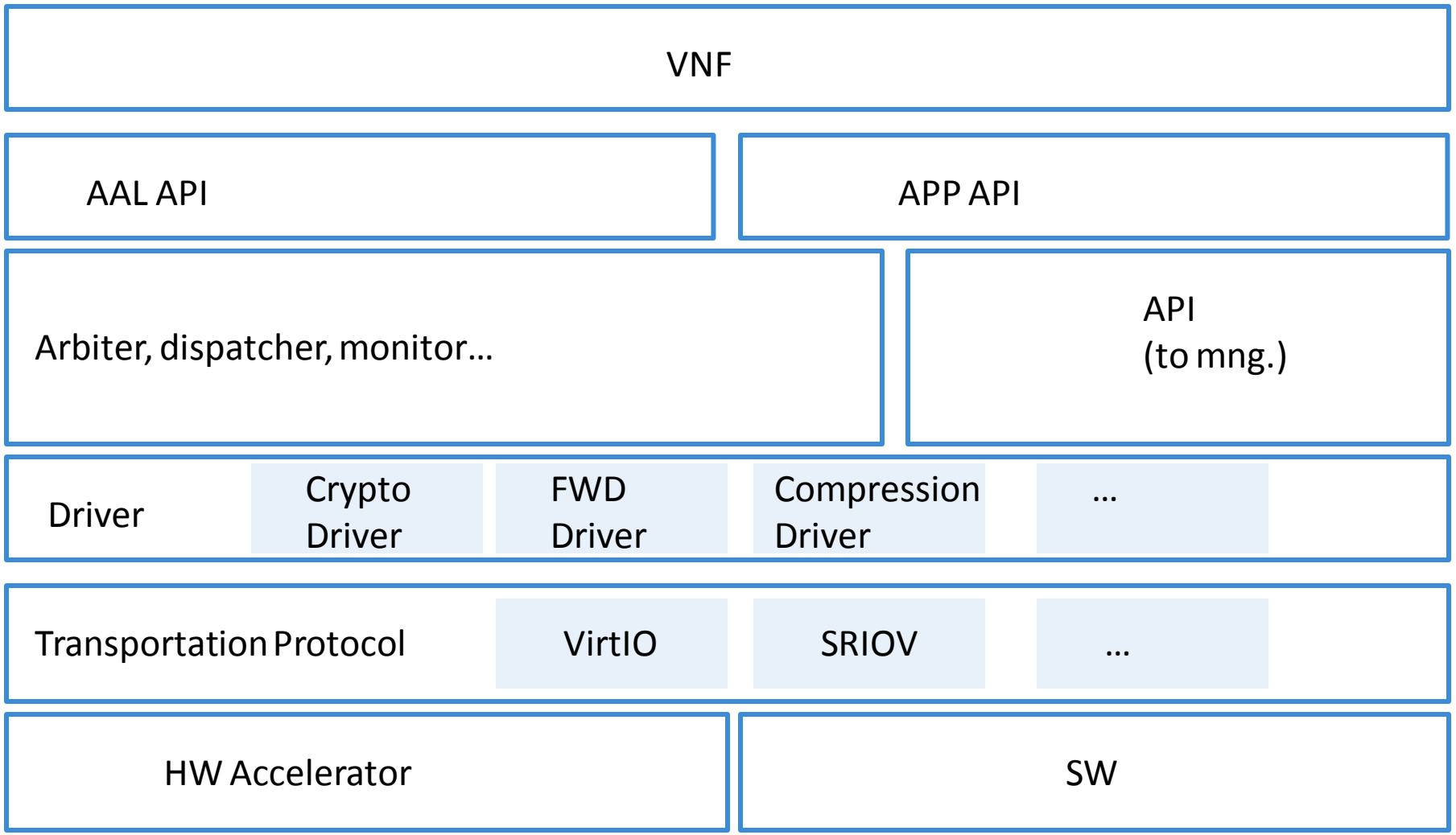
Accelerator powered IPSEC API

Framework

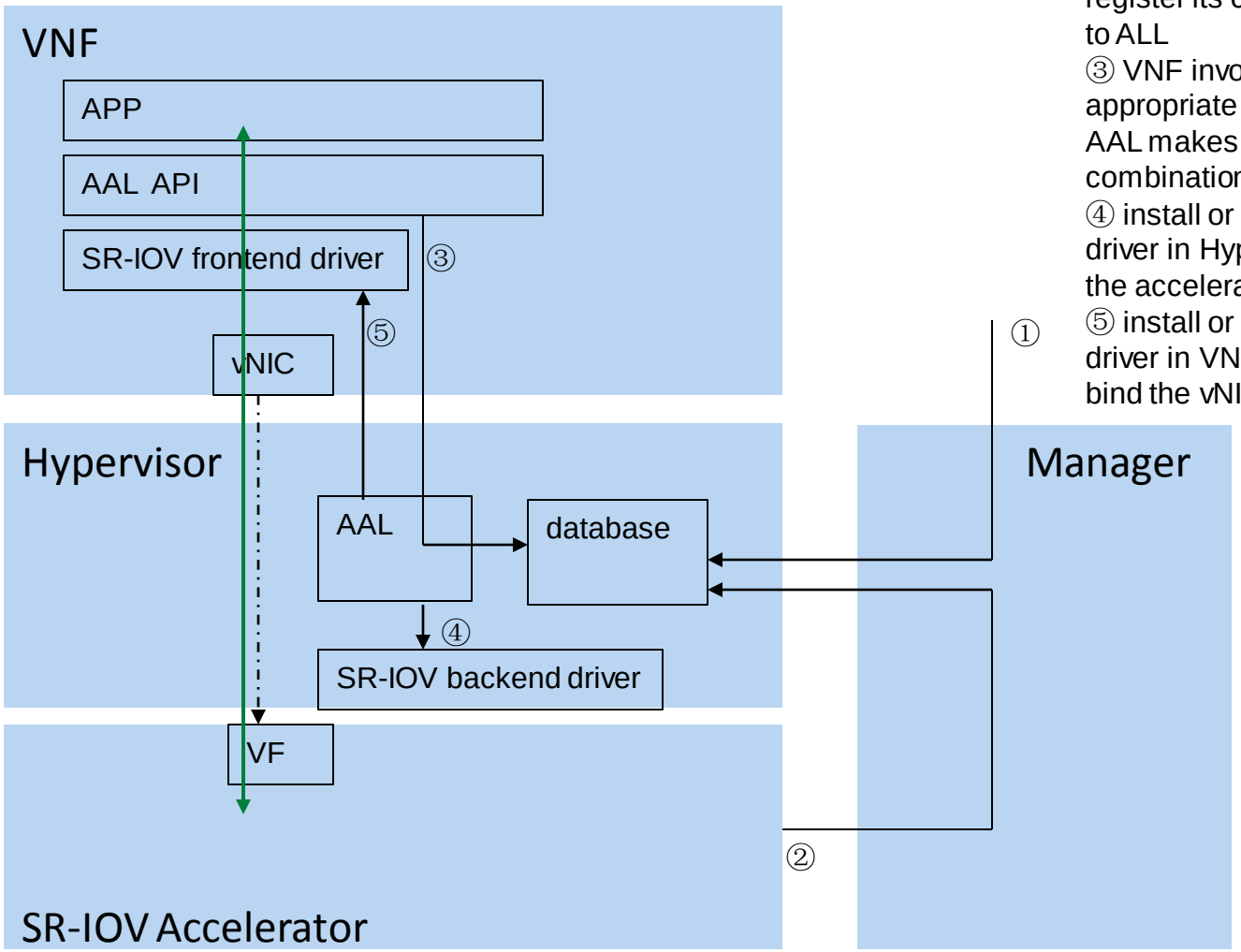


- **AAL：a set of open and common APIs, which shields the difference of accelerators as well as offer a uniform interface for VNFs. VNFs can involve these APIs to implement various of accelerated functions (e.g. Crypto, Compression, DPI and Forwarding)**
- **Manager：in charge of the management and the orchestration of accelerator resources and bridging between VNFs and accelerators.**

Framework


VNF
AAL API
APP API
Arbiter, dispatcher, monitor…
API
(to mng.)
Driver
Crypto Driver
FWD Driver
Compression Driver
…
Transportation Protocol
VirtIO
SRIOV
…
HW Accelerator
SW

Setting up for SR-IOV accelerator

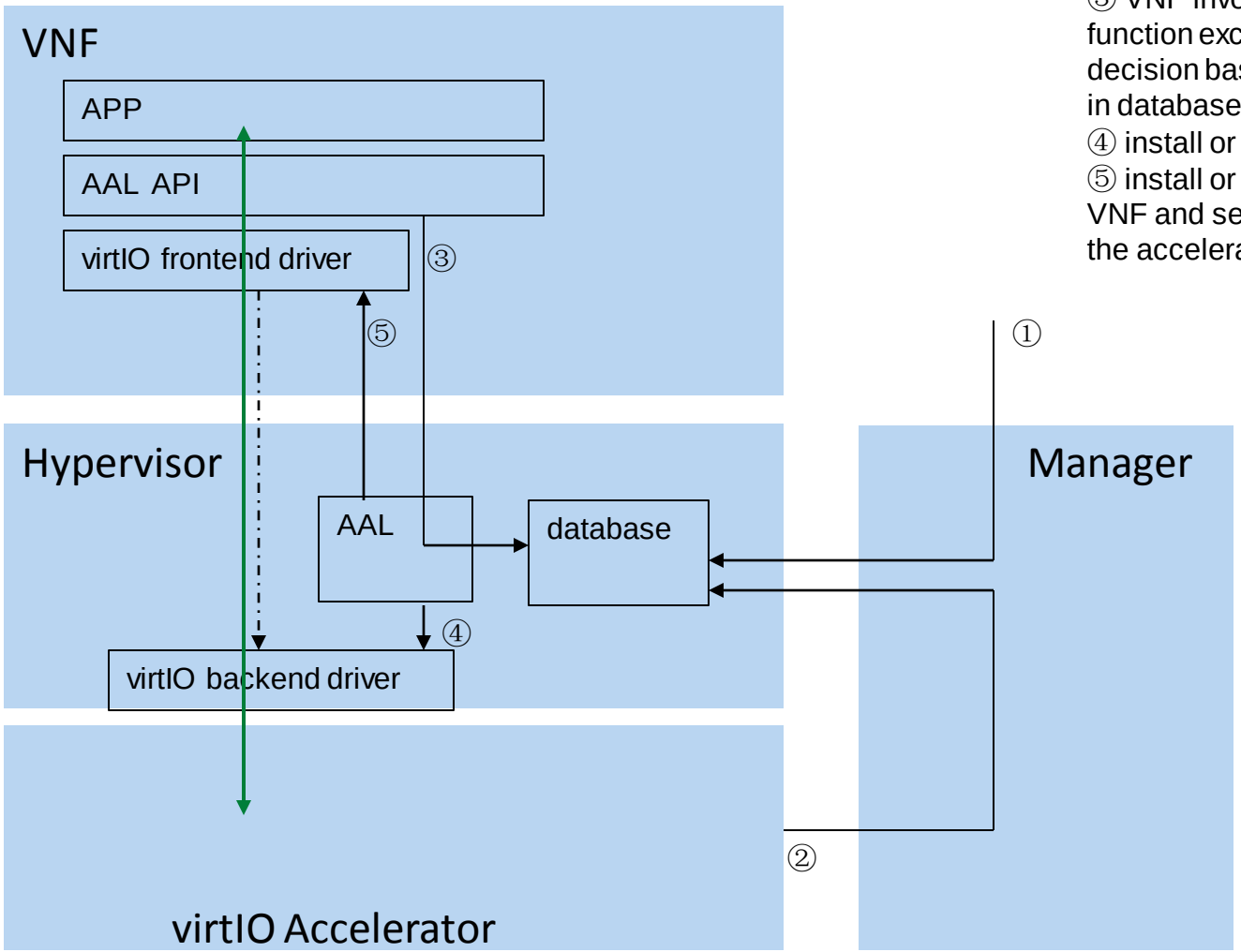


① Remote policy via manager to tell AAL whether or not the VNF will be accelerated
② Manager find accelerators and register its capability and characters to ALL
③ VNF invokes API to get an appropriate function excuser, then AAL makes the decision based on a combination of factors in database
④ install or enable SR-IOV backend driver in Hypervisor to create a VN on the accelerator.
⑤ install or enable SR-IOV frontend driver in VNF and setup it so as to bind the vNIC with VF.

↔ The data path between APP and Accelerator

Setting up for virt-IO accelerator

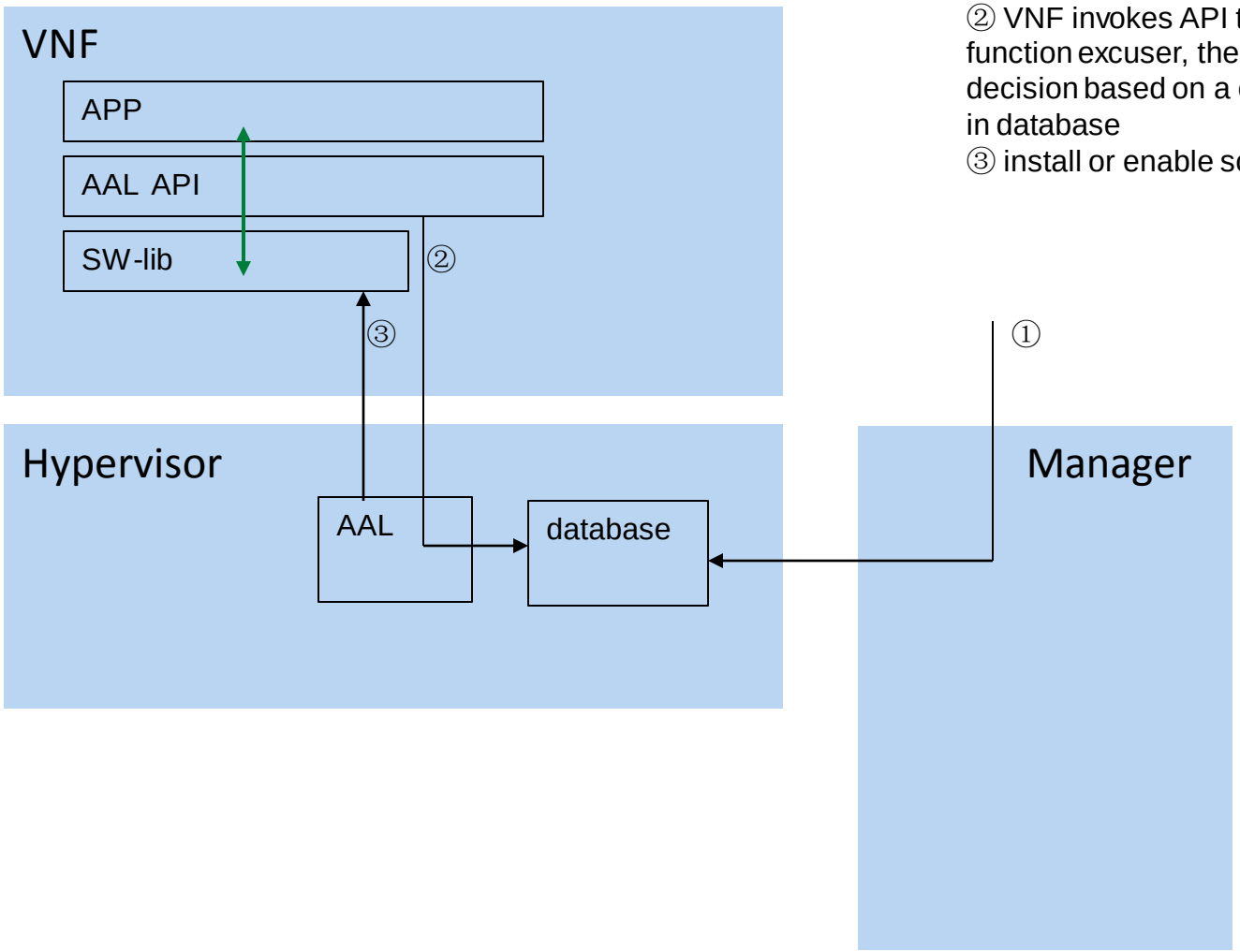
① Remote policy via manager to tell AAL whether or not the VNF will be accelerated
② Manager find accelerators and register its capability and characters to ALL
③ VNF invokes API to get an appropriate function excuser, then AAL makes the decision based on a combination of factors in database
④ install or enable virtIO backend driver in.
⑤ install or enable virtIO frontend driver in VNF and setup it so as to bind the VNF and the accelerator together.



The data path between APP and Accelerator

Setting up for none accelerator

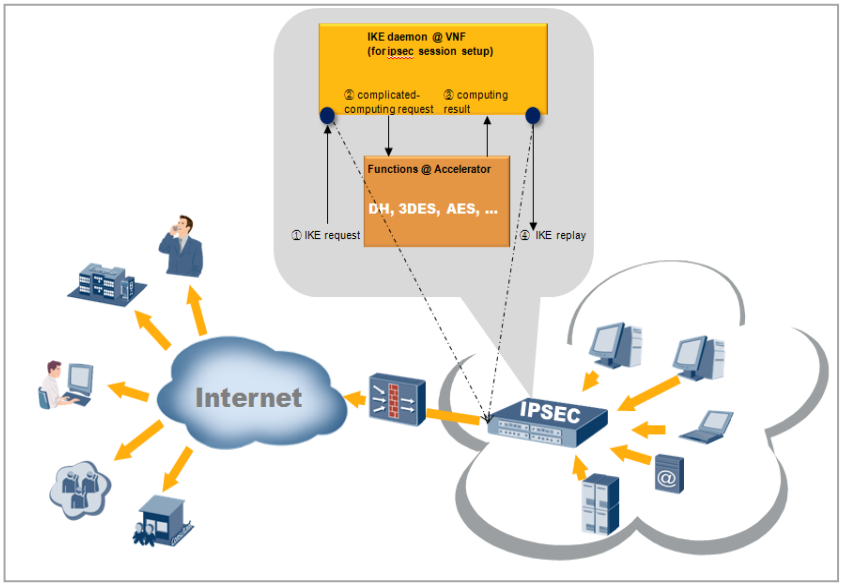
① Remote policy via manager to tell AAL whether or not the VNF will be accelerated
② VNF invokes API to get an appropriate function excuser, then AAL makes the decision based on a combination of factors in database
③ install or enable software library for VNF.



The data path between APP and software library

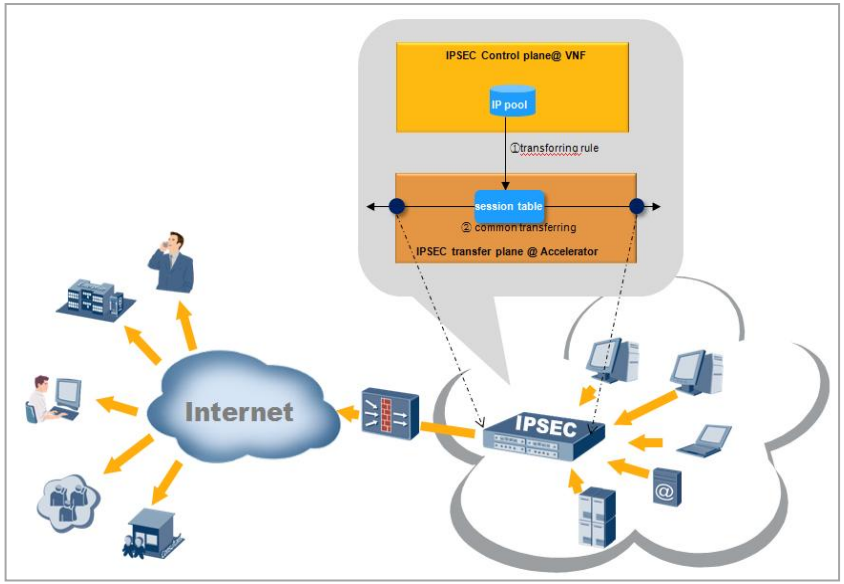
POC: accelerator-powered IPSec VPN

IKE acceleration



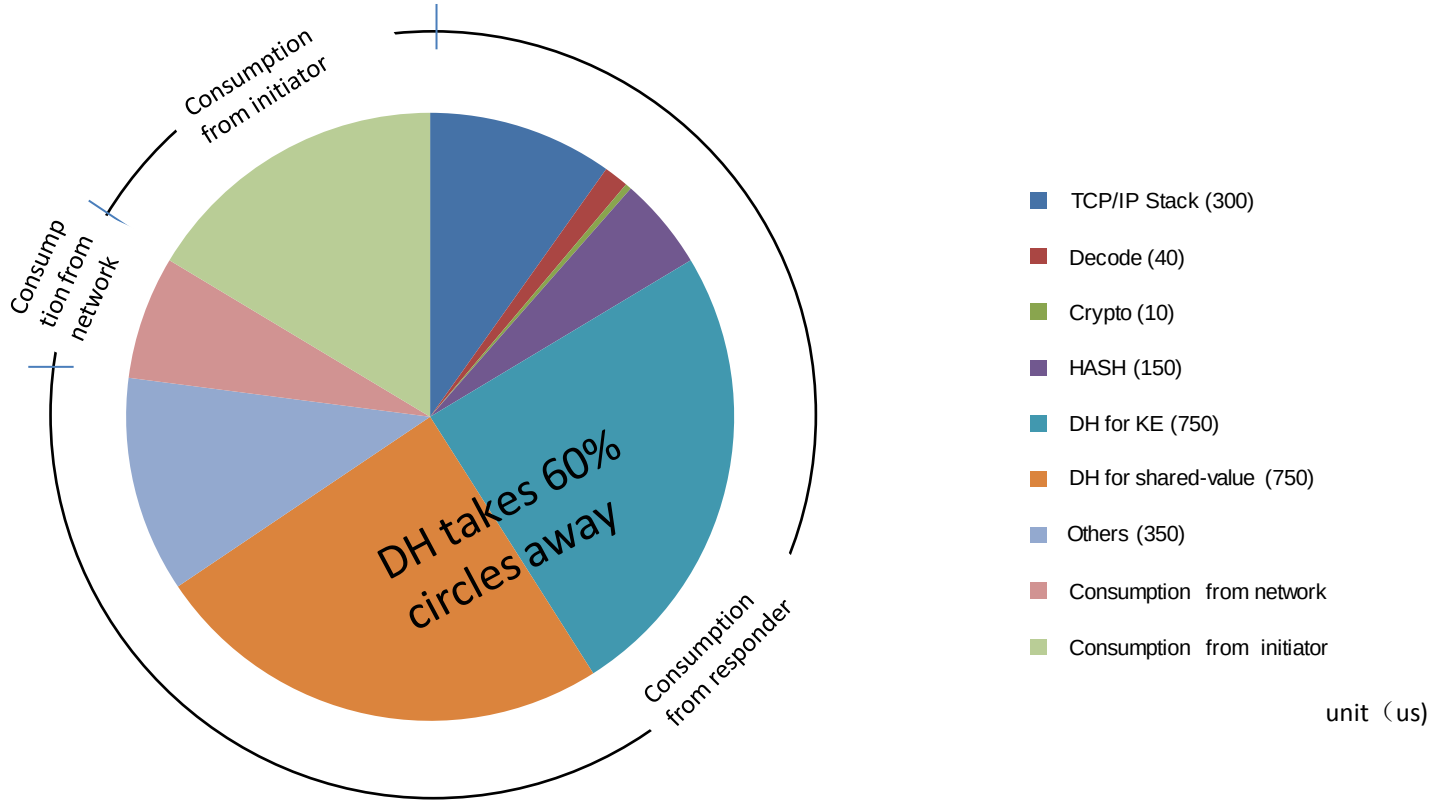
It illustrates the acceleration of IKE (IPSEC session negotiation). The DH (Differ-Hellman) function, which consumes considerable CPU resources, is the most complicated computing during IKE negotiation stage. Besides, the IPSEC gateway, which is responsible for abundant and frequent session setting-up requirements, is always expected to have a high performance.

IPSec forwarding acceleration

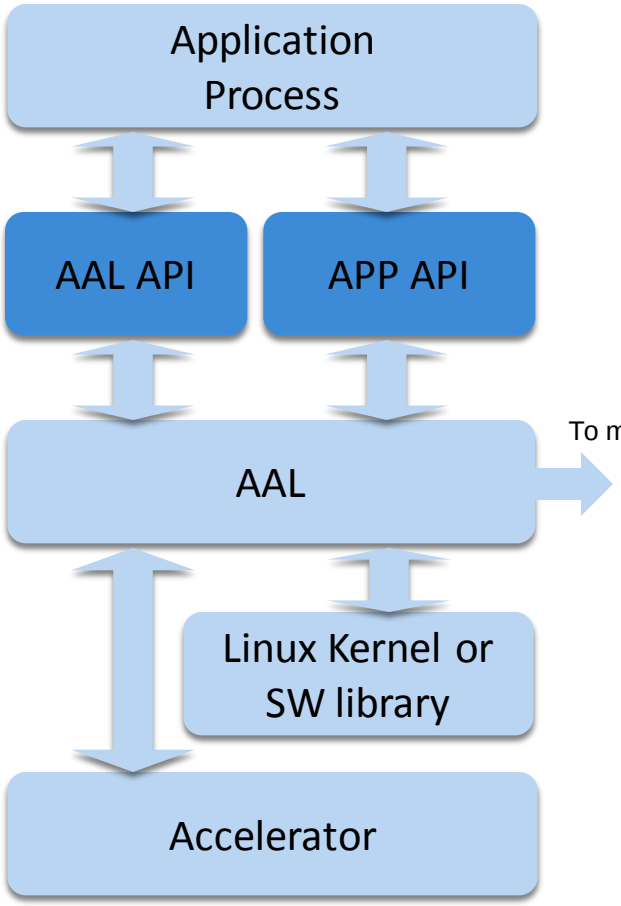


It illustrates the acceleration of IPSec forwarding. The entire forwarding-related functions, including packet encrypt/decrypt and packet RX/TX via NIC, can be offloaded accompanied with downloaded SA polices from the control-plane.

Bottleneck analysis of IKE



API infrastructure over AAL



AAL API is for runtime accelerator (or virtual accelerator) detection, allocation, binding, initiation and access.

APP API goes for application requirement.

AAL delivers requests from AAL API to accelerator directly. Meanwhile, it delivers requests from APP API to software (kernel or software library) or bound accelerator , which depends on the manager's input.

To manager

AAL API	Description
aal_context_init()	Make the environment available to VNF, such as installing the PCI driver, initiating the buffer pool, etc.
aal_find_device( )	Find out whether a proper accelerator can be used, according to the requirement input , such as the capability of the accelerator.
aal_acquire_device( )	Gain the control of the accelerator individually and initiate it (such as installing FPGA image if necessary).
aal_release_device( )	Release the control of the accelerator

APP API (Crypto API as an example)

Synchronous Crypto API	parameters	Description
aal_crypto_handle_create( )	AAL_DEVICE	Used to initialize a crypto handle. This function returns a unique handle.
aal_crypto_handle_destroy( )	CRYPTO_HANDLE	Destroy a previously initialized handle.
aal_crypto_encode( )	CRYPTO_HANDLE Input_buffer, input_buffer_len, Output_buffer, output_buffer_len	Consumes data from the input buffer and generates encrypted data in the output buffer.
aal_crypto_decode( )	CRYPTO_HANDLE Input_buffer, input_buffer_len, Output_buffer, output_buffer_len	Consumes encrypted data from the input buffer and generates decrypted data in the output buffer.

Asynchronous Crypto API	parameters	Description
aal_crypto_handle_create_async( )	AAL_DEVICE	Used to initialize an async crypto handle. This function returns a unique handle.
aal_crypto_handle_destroy_async( )	CRYPTO_HANDLE_Async	Destroy a previously initialized handle.
aal_crypto_encode_Async( )	CRYPTO_HANDLE_Async Input_buffer, input_buffer_len	Consumes data from the input buffer and when encryption finished, crypto driver call the Encode_callback function to utilize encrypted results.
aal_crypto_decode_Async( )	CRYPTO_HANDLE_Async Input_buffer, input_buffer_len	Consumes encrypted data from the input buffer and when decryption finished, crypto driver call the Decode_callback function to utilize decrypted results.

A Minimal Program

```
#include <aalh>


int main(int argc, char **argv)
{
  struct aal_context context;
  struct aal_dev *dev_1;
  struct aal_find_opt fo;
   struct crypto_handle *handle_1;
   BUFFER *input_buf, *output_buf;
   int len_in, len_out;

  aal_context_init(&context);
  dev_1 = malloc(sizeof(struct aal_dev));
  aal_device_init(&context, dev_1);
  aal_find_opt_init(&fo);

  aal_find_dev(&context, &fo, dev_1->parent.id);
  aal_acquire(dev_1);


// initialization complete, from here on regular APP API can be used
   handle_1 = aal_crypto_handle_create(dev_1);
   …
   aal_crypto_encode(handle_1, input_buf, len_in, output_buf, &len_out);
   …
   aal_crypto_handle_destroy(handle_1);

/* after using the accelerator, resources should be freed. */
  aal_release(dev_1);
  free(dev_1);

  return 0;
}
```

Thank you!