PNA - Inline Accelerators Proposal

Advanced Micro Devices

08/22/2022

AMD

Inline accelerators - Assumptions and Requirements

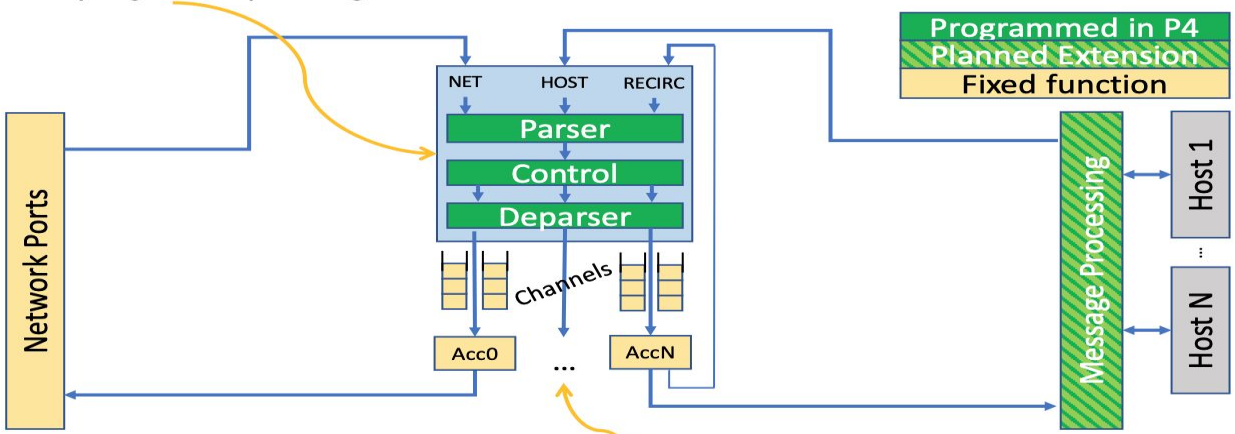
- Inline accelerators are available after each/some pipelines
- Accelerator objects and methods should abstract vendor specific implementations to provide uniform programming interface
  - Avoid definition of specific metadata (e.g. pna_pre_output_metadata_t) or headers that will vary between implementations
- Provide ability to add vendor specific extensions
- Pipeline's control and match-action functions should be able access the accelerator object and methods (May be even parser?)
- Accelerator object should provide a mechanisms to query results when applicable

Possible placement of Accelerators (from an earlier presentation)

[Public]

Proposal – 29.07.22 – Thomas Calvert

- Parser may be separate program (entry point) depending on source (NET, HOST, RECIRC).
- Control block shares state between all sources; also sets up accelerator params.
- Deparser may be separate program depending on accelerator choice.




```
Control(…) {
  …
  setup_accelerator(ACC_FOO, chan, args);
}
```

- Accelerator HW offloads are vendor-defined.
- Each may output to 1 or more destinations (NET, HOST, RECIRC).
- Targets should provide a no-op accelerator for e

AMD

Accelerator/Offload Extern Object

- Different objects specific to each class of functionality, E.g. crypto, compression, checksums, …
  - One size does not fit all
- Objects can be instantiated globally or within specific pipeline control functions
- Object methods can be used to incrementally update the information throughout the pipeline
- Multiple instances of the same object can be created

Accelerator Object Definition Example - IPSec - AES-GCM

```
extern crypto_accelerator {
   // Constructor - Can we move algorithm as constructor parameter?
   // Some methods provided in this object may be specific to the algorithm used.
   // Compiler may be able to check and warn/error when incorrect methods are used
   crypto_accelerator();  // OR crypto_accelerator(in crypto_algorithm_e algo);
   void init(crypto_algorithm_e algo);

   // security association index for this security session
   // Some implementations do not need it.. in that case this method should result in no-op i.e.
   // ignored by backend compiler
   void set_sa_index<T>(in T sa_index);

   // Set the initialization data based on the protocol used. E.g. salt, random number/ counter for ipsec
   void set_iv<T>(in T iv);
   void set_key<T,S>(in T key, in S key_size);   // 128, 192, 256
   ….
```

Object definition - continued

```
// authentication data format is protocol specific
// Add this data as a header into the packet and provide its offset and length using the
// following APIs
// The format of the auth data is not specified/mandated by this object definition
void set_auth_data_offset<T>(in T offset);
void set_auth_data_len<T>(in T len);

// Alternatively: Following API can be used to construct protocol specific auth_data and
// provide it to the engine.
void add_auth_data<T>(in T auth_data);

// Auth trailer aka ICV is added by the engine after doing encryption operation / checked after decryption
// Specify icv location - when a wire protocol wants to add ICV in a specific location (e.g. AH)
// The following apis can be used to specify the location of ICV in the packet
// special offset (TBD) indicates ICV is after the payload
void set_icv_offset<T>(in T offset);
void set_icv_len<T>(in T len);
```

Object definition - continued

```
    // setup payload to be encrypted/decrypted
    void set_payload_offset<T>(in T offset);
    void set_payload_len<T>(in T len);

    // setup the operation - the engine is assumed to perform the operation asynchronously as
    // acceleration engines are at the end of the pipeline
    void enable_encrypt<T>(in T enable_auth);
    void enable_decrypt<T>(in T enable_auth);

    // disable the engine
    void disable();

    // get results of the previous operation
    crypto_results_e get_results();
}
```

Accelerator Object Usage Example* - ipsec

```
// Instantiate accelerators
crypto_accelerator()  ipsec_acc;
crypto_accelerator()  cbc_ipsec_acc;

control IngressPipeline(inout cap_phv_intr_global_h intr_global,
                              inout cap_phv_intr_p4_h intr_p4,
                              inout ingress_headers hdr,
                              inout metadata_t metadata) {
    apply {
          ……..
        ipsec_post_decrypt_process.apply(intr_global, intr_p4,
                                  hdr, metadata);
        …..
        ipsec_sa_lookup.apply(intr_global, intr_p4, hdr, metadata);
    }
}
```

```
        action ingress_ipsec_esp_decrypt(in bit<32> spi,
                                         in bit<32> salt,
                                         in bit<256> key,
                                         in bit<9>  key_size,
                                         in bit<1>  ext_esn_en,
                                         in bit<1>  enable_auth,
                                         inout bit<64> esn) {

        ipsec_acc.init(crypto_algorithm_e.AES_GCM);

        bit<64>     seq_no;
        seq_no[31:0] = hdr.esp.seq;
        seq_no[63:32] = esn[63:32];

        // build IPSec specific IV
        bit<128> iv = salt ++ hdr.esp_iv.iv;
        ipsec_acc.set_iv(iv);
        ipsec_acc.set_key(key, key_size);

        < …. Refer to github for complete example ….>

        ipsec_acc.set_payload_offset(pyld_offset);
        ipsec_acc.set_payload_len(pyld_len);
        ipsec_acc.decrypt(enable_auth);
}
```

* Complete example will be available on github

AMD

Summary - Next Steps

- Define accelerator objects in PNA specification for known acceleration functions - checksums, crypto, compression …
- Provide example code (github) for ipsec accelerator

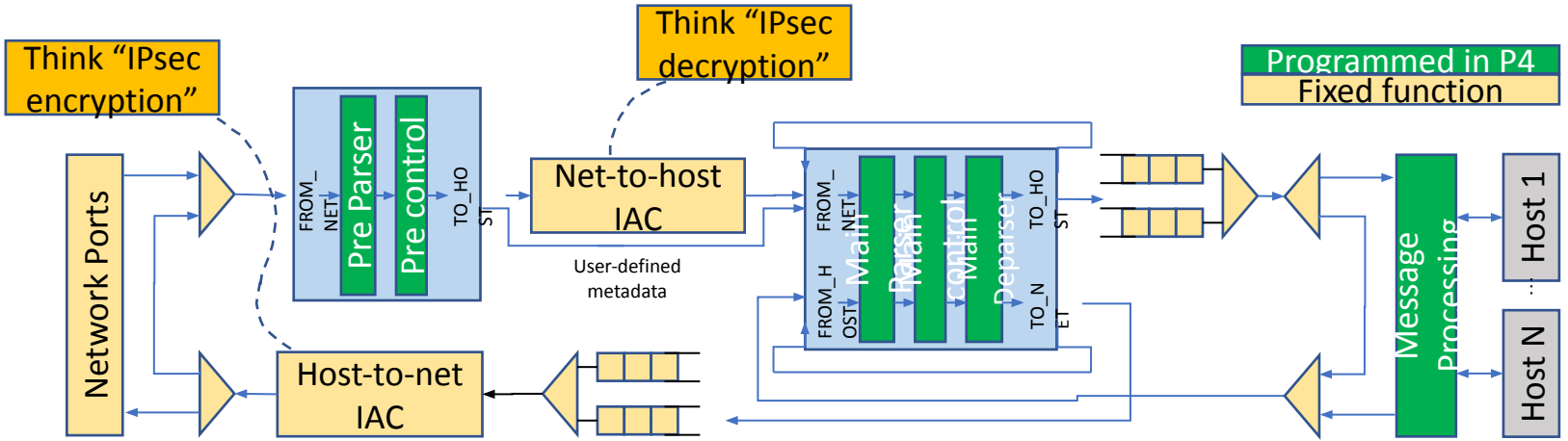
Backup slides

Extern Function vs Extern Object Comparison

Function - ipsec_acc(meta.op, meta.spi, meta.key…….)	Object - (as explained)
Parameters passed to function must be built in metadata along the pipeline. This requires 1) extra metadata allocation, 2) additional instructions to copy and reformat according to accelerator requirements	Parameters are added using object methods along the pipeline - Compiler can correctly format and adjust data in a way suitable to hardware either in metadata or by accessing accelerator directly (depending on vendor implementation). Serves as both architecture implementation hiding and vendor specific extensions.
One function that fits all cases is hard to achieve. Any add/remove of parameters will break existing code.	Vendors can add methods for their specific needs which can be ignored by other vendors who don't need it. - More flexible, object-oriented-like approach
Creating multiple instances of the same accelerator type may require additional parameters.	Easy to instantiate one or more objects based on one or more accelerators.
Function execution in control block may have to be converted to special match-action table with no keys.	Object methods will be invoked in match actions, not directly in control function.

From Earlier Presentation by Andy Fingerhut …

Architecture #2 (A2, “2 parsers”)



- As of today, this IS NOT in the public PNA specification
- We at Intel have thought about it quite a bit, and it is more difficult than we expected to compile source code written for A1 to a device whose hardware has the “shape” above.
  - Only ways found so far require either undesirable performance penalties, or unusual restrictions and difficulty in explaining to developers the way to map behavior between them.