Compression & Asymmetric Crypto in DPDK
Agenda

- What it covers?
  - DPDK compression
    - Bird – eye view
    - Key concepts & Design principle in DPDK
    - Enqueue : Example depiction
    - Roadmap
  - Asymmetric crypto in DPDK
    - What it supports? & How it support: Key design aspects & Challenges
    - Roadmap
- What it doesn’t?
  - No algorithmic specific details
  - No application specific use-case description
Compression at-a-glance

- **What we compress**
  - A lossy compression stuff
  - Lossless compression stuff

- **How do we compress**
  - Different content, Different methodologies
Compression at-a-glance contd...

• key to compression : Data DUPLICACY .. aaaaabb = 5a2b
• Algorithms refers to past data to look for repetition
  ➢ Different algorithms, Different way of encoding repetition, Different look up window size into past
Compression at-a-glance contd...

• Stateless Compression
  ➢ Each chunk independently compressed. Length of lookup window is limited to current buffer
  ➢ Decompressor doesn’t need to maintain past data

• Stateful Compression
  ➢ Each chunk compressed with reference to past data in previous buffer. Length of lookup window can slide up to limit mentioned by algorithm
  ➢ Decompressor need to maintain history buffer containing previous data
DPDK Compression

• Only for lossless. Added a new library `lib/librte_compressdev`

• Current supported algorithms:
  ➢ Deflate, LZS ONLY
  ➢ Widely used in Networking and Storage applications

• Key concepts:
  ➢ Operation types: RTE_COMP_STATELESS, RTE_COMP_STATEFUL
  ➢ Session AND Stream
    ▪ Session carry common one-time parameters
      - compress / decompress, algorithm and associated parameters: compression speed, coding etc.
      - Session types: Compression ONLY, Decompression ONLY, Compression + Hash, Decompression + Hash
      - key data structure: `rte_comp_xform`: initializes session with one-time information
DPDK Compression:

• Stream
  - Represent one data set. Example, one file is one stream
  - Carry data/operation-specific information
    - History, operational state etc
  - Stream can be one single data-chunk or broken into multiple chunks
    - Say, a very big file broken several small pieces and sent for compression
DPDK Compression: Mapping to DPDK

STREAM

C:\MyFile.exe

Compressor

Entry 1

Entry 2

C:\MyFile.zip

STREAMS

C:\file1

Compressor

Entry 1

Entry 2

C:\file1

Stateless

Stateful

To

compress
wait
compress

session

op

op

op

session

op

op

op

session

op

op

op

compress
compress
compress
DPDK Compression: Mapping to DPDK: Enqueue: Example Depiction

*Each op in a burst carry independent stream data
*Refer to Appendix Section for an Example API illustration for Setup ‘n’ Enqueue
DPDK Compression: Roadmap

• Present status
  ➢ API Patch v1 is out: https://dpdk.org/dev/patchwork/patch/34900/
    ▪ API specification with Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) PMD targeted for 18.05
  ➢ Stateless support is bare minimum requirement
    ▪ PMD can reflect its support for Stateful compression via Feature Flags
  ➢ Basic Stateless compression proof-concepted with SW ZLIB library based PMD

• Next in plan
  ➢ Plaintext Hash generation along with compression/decompression (sha-1,sha-256)
    ▪ RFC submitted by Cavium
  ➢ Preset dictionary load for stateful protocol support
  ➢ ZLIB/GZIP format support
DPDK Compression : Contributors

• RFC presented by Intel Fiona Trahe (fiona.trahe@intel.com)
• Contributed by
  - CAVIUM Shally Verma (shally.verma@caviumnetworks.com)
  - NXP Ahmed Mansour (ahmed.mansour@nxp.com)
• Joint development effort through a (short-lived) draft github repo at https://github.com/pablodelara/dpdk-draft-compressdev
DPDK Asymmetric Crypto: Overview

- Two forms of Crypto

Symmetric Encryption

- Secret Key
- Same Key
- Secret Key

Plain Text → Encryption → Cipher Text → Decryption → Plain Text

Asymmetric Encryption

- Public Key
- Different Keys
- Secret Key

Plain Text → Encryption → Cipher Text → Decryption → Plain Text
DPDK Asymmetric Crypto : Why we need it?

- Asymmetric crypto operations are **very compute-intensive**, thus HW acceleration is key advantage to applications

- DPDK asymmetric crypto exposes interface to enable Public-Private key based protocols/applications to leverage HW acceleration of part of the operations
  - Public-Private Key Generation
  - Signature Generation and Verification
  - Encryption / Decryption
    - typically used for key encryption / decryption
DPDK Asymmetric Crypto : Support

- **Extended** `lib/librte_cryptodev`

- **Current Proposal:**
  - **Session based**
    - One-time parameters on a session (algorithms & associated parameters)
  - **Asymmetric algorithms**
    - RSA Sign, Verify, Encrypt and Decrypt
    - DSA Digital Signature Generation and Verification
    - Diffie-Hellman Key Generation and Shared Secret Computation
    - Elliptic Curve DSA
    - Elliptic Diffie-Hellman Key Generation and Shared Secret Computation
    - Modular exponentiation and inversion & Fundamental Elliptic Curve operations
DPDK Asymmetric Crypto contd...

• Key Design Challenge
  ➢ Different vendors, Different crypto capable devices. They can support
    ▪ **Either** symmetric or asymmetric only
      - Device and all of its queue pair support only one type of crypto operation
    ▪ **Or, Both** symmetric + asymmetric
      - Device and all of its qp support both of crypto operations
DPDK Asymmetric Crypto contd...

• Or a third kind of crypto device that
  ➢ Support both symmetric + asymmetric but have dedicated queues per each crypto type
    ▪ Such devices can split themselves into Symmetric ONLY and Asymmetric ONLY instances
DPDK Asymmetric Crypto : Usage

• For PMD that support both Symmetric (s) And Asymmetric (A),
  ➢ Most generic expected usage on queue pair (qp):

<table>
<thead>
<tr>
<th>qp0</th>
<th>S</th>
<th>S</th>
<th>S</th>
<th>S</th>
<th>S</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>qp1</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

  - Application will enqueue one service type at-a-time on a qp

  - Is it a requirement?!
DPDK Asymmetric Crypto : Roadmap

• Present status
  - RFC patch on dpdk-next-crypto submitted and open for feedback
    https://dpdk.org/dev/patchwork/patch/34308/
  - Proof-conceived proposal with Openssl SW PMD (as Symmetric + Asymmetric capable)
    - Partly tested for RSA and Modular operations

• Next in plan
  - Targeting patch v1 by End-March
DPDK Asymmetric Crypto : Contributors

- RFC initiated by Umesh Kartha, umesh.kartha@caviumnetworks.com
  - Continued by Shally Verma, shally.verma@caviumnetworks.com

- Contributed by Fiona Trahe, fiona.trahe@intel.com
Appendix: DPDK Compression Example API

Illustration: Setup n Enqueue single op for stateless compression
Appendix : DPDK Compression Example API illustration contd...

```
rte_compress_dev_info_get(dev_id, dev_info_ptr)

---

dev->dev_ops->dev_infos_get(dev, dev_info_ptr)

---

updated dev_info_ptr, nb_qp

---

rte_compress_dev_config(dev_id, dev_config[nb_qp])

---

dev->dev_ops->dev_configure(dev, nb_qp)

---

pass/fail

---

rte_comp_stream_create(dev_id, comp_sess, (void **)stream_ptr, op_type)

---

dev_ops->stream_create((dev, comp_sess, void **stream_ptr, op_type)

---

valid stream_ptr (opaque to user)
```
Appendix : DPDK Compression Example API illustration contd...

```
rte_comp_op_pool_create(name, nb_ops...)

rte_mempool *op_pool

rte_op.bulk_alloc(op_pool, ops, nb_ops)

rte_compressdev_queue_pair_setup(devid,qpid,max.inflight_ops...)

dev->dev_ops->queue_pair_setup(dev,..)

rte_compressdev_start(devid)

dev->dev_ops->dev_start(dev)
```
Appendix: DPDK Compression Example API illustration contd...

```
app_setup op, then enqueue
/* allocate op */
rtc_comp_op *op = rtc_comp_op_alloc(0, op_size, caller); // Allocate DPDK compression operation structure
/* attach session and stream to op */
op->session = session;
op->stream = stream;
/* setup data buffer pointers */
op->m = src;
op->src.offset = 0;
op->src.len = src_len;
op->m.start = cbaf;
op->dst.offset = 0;
op->flush_flag = RTE_COMP_FLUSH_FINAL;
op->type = RTE_COMP_OP_STATELESS;

rtc_compressdev_enqueue_burst(devid, op, nb_ops)

push_to_compressor
```

loop until an op is dequeued
THANK YOU

ThunderX, XPIant, OCTEON, OCTEON Fusion, NITROX, LiquidIO are registered trademarks of Cavium., OCTEON TX, LiquidSecurity are trademarks of Cavium.