

# Assignment 3 report: DPU 网络编程

信息科学技术学院本科生：王子骁 毛川 刘智琦 王兴辰

2025 年 6 月 18 日

## 1 Secure Channel

### 1.1 DOCA secure channel 复现

我们需要在主机和 DPU 之间创建安全通信通道，Host 会向 DPU 发送连接请求，DPU 接受连接。之后 Host 会向 DPU 发送 message, DPU 接收到之后会给出 response。我们按照指示 build 了 doca secure channel, 之后我们进行了如下操作：

在 DPU 上，我们通过运行

`doca_caps --list-rep-dev`

命令，查找到了 DOCA Secure Channel device PCIe address 和 DOCA Comm Channel device representor PCIe address，分别是 0000:03:00.0 和 0000:18:00.0，然后我们在 DPU 端和 Host 端分别执行下面的命令，成功建立连接并发送了 10 条字节大小为 256 的消息：

```
ubuntu@localhost:/tmp/build/secure_channel$ ./doca_secure_channel -s 256 -n 10 -p 0000:03:00.0 -r 0000:18:00.0
[06:11:28:608099][2041][DOCA][INF][secure_channel_core.c:491][init_cc] Started Listening, waiting for new connection
[06:12:34:143966][2042][DOCA][INF][secure_channel_core.c:233][sendto_channel] Send thread exiting, total amount of messages sent successfully: 10
```

图 1: 复现结果 dpu

```
nvidia@doca03:/tmp/build/secure_channel$ ./doca_secure_channel -s 256 -n 10 -p 0000:03:00.0
[06:12:34:818711][97592][DOCA][INF][secure_channel_core.c:491][init_cc] Started Listening, waiting for new connection
[06:12:34:818869][97594][DOCA][INF][secure_channel_core.c:233][sendto_channel] Send thread exiting, total amount of messages sent successfully: 10
```

图 2: 复现结果 host

可以看到，尽管输出的信息比较少，但是 DPU 和 Host 之间的连接已经成功建立，10 条消息成功被发送和接收。表明 Doca secure channel 建立成功。

### 1.2 修改 msg-size

在这里我们修改了几个 msg-size 的参数，情况如图 我们发现当 msg-size 大到一定值之后就会报错，

```
ubuntu@localhost:~$ cd /tmp/build/secure_channel
ubuntu@localhost:/tmp/build/secure_channel$ ./doca_secure_channel -s 256 -n 10 -p 0000:03:00.0 -r 0000:18:00.0
[06:28:53:612051][15092][DOCA][INF][secure_channel_core.c:491][init_cc] Started Listening, waiting for new connection
[06:30:37:425295][15093][DOCA][INF][secure_channel_core.c:233][sendto_channel] Send thread exiting, total amount of messages sent successfully: 10
```

图 3: size=256

```
ubuntu@localhost:/tmp/build/secure_channel$ ./doca_secure_channel -s 4096 -n 10 -p 0000:03:00:0 -r 0000:18:00:0
[06:35:40:215843][20322][DOCA][ERR][secure_channel_core.c:74][message_size_callback] Received message size is not supported
[06:35:40:215953][20322][DOCA][ERR][doca_argp.cpp:849][parse_cli_arguments] Encountered an error [Invalid input] while handling param "s"
Encountered an error [Invalid input] while handling param "s"

Usage: doca_secure_channel [DOCA Flags] [Program Flags]

DOCA Flags:
-h, --help                Print a help synopsis
-v, --version             Print program version information
-l, --log-level           Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
--sdk-log-level           Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
-j, --json <path>        Parse all command flags from an input json file

Program Flags:
-s, --msg-size            Message size to be sent
-n, --num-msgs           Number of messages to be sent
-p, --pci-addr           DOCA Comm Channel device PCI address
-r, --rep-pci            DOCA Comm Channel device representor PCI address (needed only on DPU)

[06:35:40:216012][20322][DOCA][ERR][secure_channel_core.c:71][main] Failed to parse application input: Invalid input
```

图 4: size=4069

这是因为在在 secure\_channel\_core.c 中定义了 CC\_MAX\_MSG\_SIZE=4080 , 所以超过 4080 就会报错, 所以只要保证在 4080 以内就可以了

### 1.3 修改 num-msgs

```
ubuntu@localhost:/tmp/build/secure_channel$ ./doca_secure_channel -s 256 -n 100 -p 0000:03:00:0 -r 0000:18:00:0
[06:38:30:644712][22585][DOCA][INF][secure_channel_core.c:491][init_cc] Started Listening, waiting for new connection
[06:38:41:012219][22586][DOCA][INF][secure_channel_core.c:233][sendto_channel] Send thread exiting, total amount of messages sent successfully: 100
^C[06:39:15:552314][22587][DOCA][INF][secure_channel_core.c:300][recvfrom_channel] Receive thread exiting, total amount of messages received successfully: 200
ubuntu@localhost:/tmp/build/secure_channel$ ./doca_secure_channel -s 256 -n 400 -p 0000:03:00:0 -r 0000:18:00:0
[06:39:23:773441][23309][DOCA][INF][secure_channel_core.c:491][init_cc] Started Listening, waiting for new connection
[06:39:28:330722][23311][DOCA][INF][secure_channel_core.c:233][sendto_channel] Send thread exiting, total amount of messages sent successfully: 400
^C[06:39:58:817634][23312][DOCA][INF][secure_channel_core.c:300][recvfrom_channel] Receive thread exiting, total amount of messages received successfully: 200
```

图 5: Host

```
nvidia@doca03:/tmp/build/secure_channel$ ./doca_secure_channel -s 256 -n 200 -p 0000:0b:00:0
[06:38:41:410934][67556][DOCA][INF][secure_channel_core.c:480][init_cc] Connection to DPU was established successfully
[06:38:41:411213][67557][DOCA][INF][secure_channel_core.c:233][sendto_channel] Send thread exiting, total amount of messages sent successfully: 200
^C[06:39:13:204486][67558][DOCA][INF][secure_channel_core.c:300][recvfrom_channel] Receive thread exiting, total amount of messages received successfully: 100
nvidia@doca03:/tmp/build/secure_channel$ ./doca_secure_channel -s 256 -n 200 -p 0000:0b:00:0
[06:39:28:743294][67846][DOCA][INF][secure_channel_core.c:480][init_cc] Connection to DPU was established successfully
[06:39:28:743616][67847][DOCA][INF][secure_channel_core.c:233][sendto_channel] Send thread exiting, total amount of messages sent successfully: 200
^C[06:39:55:379230][67848][DOCA][INF][secure_channel_core.c:300][recvfrom_channel] Receive thread exiting, total amount of messages received successfully: 400
```

图 6: Dpu

在这里我们修改两边的的 num-msgs 数量不同, 比如说输出 200 条消息, 如果接受只设置为 100 , 就只会接受 100 条消息, 反之发送 100 条我接收端设置为 200 也只会接受 100 条消息

## 1.4 Log

为了验证日志系统的有效性，我们分别在程序正常运行和异常运行两种场景下，测试了不同日志等级 (Log Level) 的输出表现。

### 1.4.1 正常运行时的详细日志 (TRACE Level)

当 ‘log-level’ 设置为 10 或 20 时，因为日志等级太低，所以输出空白

```
ubuntu@localhost:/tmp/build/secure_channel$ ./doca_secure_channel --json /opt/mellanox/doca/applications/secure_channel/sc_params.json
```

图 7: 10

```
ubuntu@localhost:/tmp/build/secure_channel$ ./doca_secure_channel --json /opt/mellanox/doca/applications/secure_channel/sc_params.json
```

图 8: 20

当 ‘log-level’ 设置为 70 (TRACE) 时，程序会输出最为详尽的运行日志。这对于理解程序的完整执行流程和进行性能分析至关重要。如下日志片段所示，系统清晰地记录了从连接建立、收发数据计时、消息传输（精确到每一条消息的发送状态）到最终成功退出的全过程。时间戳精度达到微秒级，为性能瓶颈分析提供了有力的数据支持。

```
^Cubuntu@localhost:/tmp/build/secure_channel$ ./doca_secure_channel --json /opt/mellanox/doca/applications/secure_channel/sc_params.json
[13:02:30:673893][309991][DOCA][INF][secure_channel_core.c:563][init_cc] Started Listening, waiting for new connection
[13:02:33:767997][309995][DOCA][INF][secure_channel_core.c:322][recvfrom_channel] Connection has been established
[13:02:33:768044][309995][DOCA][INF][secure_channel_core.c:323][recvfrom_channel] RECV START TIME: 1148451.355258098 seconds
[13:02:33:768071][309995][DOCA][INF][secure_channel_core.c:331][recvfrom_channel] receive end at: 1148451.355332939 seconds
[13:02:33:768100][309994][DOCA][INF][secure_channel_core.c:214][sendto_channel] SEND START TIME: 1148451.355361642 seconds
[13:02:33:768146][309994][DOCA][INF][secure_channel_core.c:216][sendto_channel] Starting message transmission. Will send 10 messages of size 512 bytes
[13:02:33:768154][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 0
[13:02:33:768166][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 1
[13:02:33:768173][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 2
[13:02:33:768180][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 3
[13:02:33:768188][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 4
[13:02:33:768195][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 5
[13:02:33:768202][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 6
[13:02:33:768209][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 7
[13:02:33:768216][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 8
[13:02:33:768223][309994][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 9
[13:02:33:768230][309994][DOCA][INF][secure_channel_core.c:266][sendto_channel] send end at: 1148451.355491798 seconds
[13:02:33:768234][309994][DOCA][INF][secure_channel_core.c:271][sendto_channel] Send thread exiting, total amount of messages sent successfully: 10
```

图 9: 70

### 1.4.2 错误处理与调试分析 (ERROR vs. DISABLE Level)

为了检验日志系统在错误排查中的价值，我们通过提供一个不支持的 ‘msg-size’ 参数来故意触发程序错误，并比较了不同日志等级下的输出。

**Log Level: 30 (ERROR)** 当 ‘log-level’ 设置为 30 (ERROR) 时，程序能够输出具体且有价值的错误信息。如下所示，日志明确指出了错误源于 `secure_channel.c` 文件的第 76 行，原因是 “Received message size is not supported”。这条精准的定位信息极大地帮助开发者快速找到问题根源。

```
14     "doca_general_flags": {
15         // -l - Set the (numeric) Log Level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG>
16         "log-level": 30
17     },
18     "doca_program_flags": {
19         // -s - message size in bytes
20         "msg-size": 10000,
21         // -n - number of messages to send
22         "num-msgs": 10,
23         // -p - commm channel doca device pci address
24         "pci-addr": "0000:03:00.0",
25     }
26 }
```

Problems Output Debug Console Terminal Ports bash - secure\_channel

-p, --pci-addr DOCA Comm Channel device PCI address  
-r, --rep-pci DOCA Comm Channel device representor PCI address (needed only on DPU)

ubuntu@localhost:/tmp/build/secure\_channel\$ ./doca\_secure\_channel --json /opt/mellanox/doca/applications/secure\_channel/sc\_params.json  
[13:08:14:835460][314994][DOCA][ERR][secure\_channel\_core.c:76][message\_size\_callback] Received message size is not supported  
[13:08:14:835553][314994][DOCA][ERR][doca\_argp.cpp:544][handle\_json\_args] Encountered an error [Invalid input] while handling param "msg-size"  
Encountered an error [Invalid input] while handling param "msg-size"

Usage: doca\_secure\_channel [DOCA Flags] [Program Flags]

DOCA Flags:  
-h, --help Print a help synopsis  
-v, --version Print program version information  
-l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
-j, --json <path> Parse all command flags from an input json file

Program Flags:  
-s, --msg-size Message size to be sent  
-n, --num-msgs Number of messages to be sent  
-p, --pci-addr DOCA Comm Channel device PCI address  
-r, --rep-pci DOCA Comm Channel device representor PCI address (needed only on DPU)

[13:08:14:835616][314994][DOCA][ERR][secure\_channel.c:80][main] Failed to parse application input: Invalid input

图 10: error 30

**Log Level: 10 (DISABLE)** 当 ‘log-level’ 设置为 10 (DISABLE) 时，应用程序自身的日志被禁止。此时，虽然程序依旧会因参数错误而退出，但我们只能看到来自 DOCA 框架底层（如 ‘doca\_argp.c’ 参数解析模块）的通用错误提示。日志仅模糊地指出在处理 ‘msg-size’ 参数时遇到 “Invalid input”，但缺失了上层应用（‘secure\_channel.c’）中关于 “消息大小不被支持” 的关键上下文信息。

```
13 {
14     "doca_general_flags": {
15         // -l - Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG>
16         "log-level": 10
17     },
18     "doca_program_flags": {
19         // -s - message size in bytes
20         "msg-size": 10000,
21         // -n - number of messages to send
22         "num-msgs": 10,
23         // -p - commm channel doca device pci address
24         "pci-addr": "0000:03:00.0",
25     }
26 }
```

Problems Output Debug Console Terminal Ports bash - secure\_channel

-n, --num-msgs Number of messages to be sent  
-p, --pci-addr DOCA Comm Channel device PCI address  
-r, --rep-pci DOCA Comm Channel device representor PCI address (needed only on DPU)

[13:05:06:716213][311882][DOCA][ERR][secure\_channel.c:80][main] Failed to parse application input: Invalid input  
ubuntu@localhost:/tmp/build/secure\_channel\$ ./doca\_secure\_channel --json /opt/mellanox/doca/applications/secure\_channel/sc\_params.json  
[13:06:24:077673][312968][DOCA][ERR][doca\_argp.cpp:544][handle\_json\_args] Encountered an error [Invalid input] while handling param "msg-size"  
Encountered an error [Invalid input] while handling param "msg-size"

Usage: doca\_secure\_channel [DOCA Flags] [Program Flags]

DOCA Flags:  
-h, --help Print a help synopsis  
-v, --version Print program version information  
-l, --log-level Set the (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
--sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>  
-j, --json <path> Parse all command flags from an input json file

Program Flags:  
-s, --msg-size Message size to be sent  
-n, --num-msgs Number of messages to be sent  
-p, --pci-addr DOCA Comm Channel device PCI address  
-r, --rep-pci DOCA Comm Channel device representor PCI address (needed only on DPU)

图 11: error 10

**对比分析**对比两种等级下的错误输出，可以明显看出，一个设计良好的分级日志系统在调试过程中起着决定性作用。‘ERROR’等级提供了定位问题所需的精确上下文，而‘DISABLE’等级虽然在生产环境中可以减少性能开销，但在问题排查上效率低下。

## 1.5 实现基于 JSON 的配置与增强的日志输出

为了提升原‘secure\_channel’程序的可配置性、易用性与可观测性，本项目实现了两大核心改进：引入 JSON 文件作为配置输入，并对日志输出系统进行了增强。

### 1.5.1 基于 JSON 文件的参数化配置

原程序依赖于命令行参数或硬编码进行配置，在进行多组参数的测试时不便且易错。为解决此问题，我们对程序源码（特别是 `secure_channel_core.c`）进行了修改，集成了 JSON 解析功能。

现在，程序可以接受一个外部 JSON 文件作为输入（如 `sc_params.json`），并从中读取运行所需的全部参数，例如消息的大小、发送的消息数量以及客户端/服务器模式等。如截图 `json.png` 所示，程序成功地从 JSON 文件中解析了配置（例如，传输 256 条大小为 128 字节的消息）并正确执行。这一改进将配置与逻辑解耦，极大地提高了实验的灵活性与效率。

```
ubuntu@localhost: /tmp/build/secure_channel$ ./doca_secure_channel --json /opt/mellanox/doca/applications/secure_channel/sc_params.json
[12:34:53:615795][284640][DOCA][INF][secure_channel_core.c:563][init_cc] Started Listening, waiting for new connection
[12:35:40:936069][284643][DOCA][INF][secure_channel_core.c:322][recvfrom_channel] Connection has been established
[12:35:40:936108][284643][DOCA][INF][secure_channel_core.c:323][recvfrom_channel] RECV START TIME: 1146839.223858631 seconds
[12:35:40:936115][284643][DOCA][INF][secure_channel_core.c:327][recvfrom_channel] Received message is:
[12:35:40:936139][284643][DOCA][INF][secure_channel_core.c:327][recvfrom_channel] Received message is:
[12:35:40:936144][284643][DOCA][INF][secure_channel_core.c:327][recvfrom_channel] Received message is:
[12:35:40:936147][284643][DOCA][INF][secure_channel_core.c:327][recvfrom_channel] Received message is:
[12:35:40:936151][284643][DOCA][INF][secure_channel_core.c:327][recvfrom_channel] Received message is:
[12:35:40:936155][284643][DOCA][INF][secure_channel_core.c:327][recvfrom_channel] Received message is:
[12:35:40:936162][284643][DOCA][INF][secure_channel_core.c:327][recvfrom_channel] Received message is:
[12:35:40:936168][284643][DOCA][INF][secure_channel_core.c:327][recvfrom_channel] Received message is:
[12:35:40:936174][284643][DOCA][INF][secure_channel_core.c:327][recvfrom_channel] Received message is:
[12:35:40:936183][284642][DOCA][INF][secure_channel_core.c:214][sendto_channel] SEND START TIME: 1146839.223973394 seconds
[12:35:40:936235][284642][DOCA][INF][secure_channel_core.c:216][sendto_channel] Starting message transmission. Will send 256 messages of
size 128 bytes
[12:35:40:936242][284642][DOCA][INF][secure_channel_core.c:220][sendto_channel] send message 0

[12:35:40:936251][284642][DOCA][INF][secure_channel_core.c:220][sendto_channel] send message 1

[12:35:40:936256][284642][DOCA][INF][secure_channel_core.c:220][sendto_channel] send message 2

[12:35:40:936263][284642][DOCA][INF][secure_channel_core.c:220][sendto_channel] send message 3

[12:35:40:936270][284642][DOCA][INF][secure_channel_core.c:220][sendto_channel] send message 4
```

图 12: json

### 1.5.2 增强的日志与性能指标输出

为了更精确地进行调试和性能分析，我们对程序的标准输出进行了增强。新的日志系统现在可以提供更丰富的运行时信息，主要包括：

- **精确计时：**记录了数据接收（RECV）和发送（SEND）阶段的纳秒级精度的起始与结束时间戳。
- **明确的传输状态：**在数据传输开始时，会明确打印出待发送消息的总数与单条消息的大小。
- **总结性报告：**在程序执行完毕后，会输出本次运行成功传输的消息总数，提供了一个清晰、直接的结果验证。

如截图 `enhanced.png` 所示，增强后的输出（例如，`send end at: ...` 和 `total amount of messages sent successfully: 10`）使得程序执行过程一目了然，为后续的性能调优和结果分析提供了可靠的数据支持。

```

ubuntu@localhost:/tmp/build/secure_channel$ ./doca_secure_channel --json /opt/mellanox/doca/applications/secure_channel/sc_params.json
[12:56:33:576814][304988][DOCA][INF][secure_channel_core.c:563][init.cc] Started Listening, waiting for new connection
[12:56:40:422933][304997][DOCA][INF][secure_channel_core.c:322][recvfrom_channel] Connection has been established
[12:56:40:422974][304997][DOCA][INF][secure_channel_core.c:323][recvfrom_channel] RECV START TIME: 1148098.010193823 seconds
[12:56:40:423011][304997][DOCA][INF][secure_channel_core.c:331][recvfrom_channel] receive end at: 1148098.010273387 seconds
[12:56:40:423032][304996][DOCA][INF][secure_channel_core.c:214][sendto_channel] SEND START TIME: 1148098.010293977 seconds
[12:56:40:423077][304996][DOCA][INF][secure_channel_core.c:216][sendto_channel] Starting message transmission. Will send 10 messages of
size 512 bytes
[12:56:40:423085][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 0
[12:56:40:423092][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 1
[12:56:40:423096][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 2
[12:56:40:423101][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 3
[12:56:40:423105][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 4
[12:56:40:423110][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 5
[12:56:40:423114][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 6
[12:56:40:423118][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 7
[12:56:40:423123][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 8
[12:56:40:423127][304996][DOCA][INF][secure_channel_core.c:220][sendto_channel] message sent: 9
[12:56:40:423131][304996][DOCA][INF][secure_channel_core.c:266][sendto_channel] send end at: 1148098.010392812 seconds
[12:56:40:423134][304996][DOCA][INF][secure_channel_core.c:271][sendto_channel] Send thread exiting, total amount of messages sent succe
ssfully: 10

```

图 13: enhanced

## 1.6 探究 message 的字节 size 和 message 数量对于发送总时间和每个消息的平均发送时间的影响

我们首先探究了在发送 10 个 message 的情形下，不同的 message 字节大小对发送时间的影响，结果如下：（每一组实验的结果图可以在 **msg size** 文件夹下保存）

表 1: 消息长度对通信时间的影响

| 消息长度（字节） | 总通信时间（微秒） | 平均每个消息通信时间（微秒） |
|----------|-----------|----------------|
| 128      | 90        | 9              |
| 256      | 90        | 9              |
| 512      | 96        | 9.6            |
| 1024     | 103       | 10.3           |

可以看到，随着消息字节大小增大，平均消息通信时间略有增加但差别不大。我们认为背后的原因是 DPU-Host 通信时间主要由固定开销主导，而非数据传输本身。从 128 字节到 256 字节消息长度翻倍时，总通信时间保持不变 (90 s)，说明在小数据量范围内，协议处理、DMA 设置等固定成本远大于实际数据传输成本。只有当消息长度增加到 512 字节以上时，数据传输时间才开始显现轻微影响，但增幅仍然很小，表明 PCIe 等高速接口的带宽利用率还远未饱和。

之后，我们探究了在发送信息的字节大小固定为 256 的情况下，消息的数量对发送总时间和平均每个消息发送时间的影响（每一组实验的结果图可以在 **msg num** 文件夹下面找到）：



表 2: 消息数量对通信时间的影响

| 消息数量 | 总通信时间（微秒） | 平均每个消息通信时间（微秒） |
|------|-----------|----------------|
| 1    | 57        | 57             |
| 2    | 62        | 31             |
| 5    | 73        | 14.6           |
| 10   | 90        | 10.3           |
| 50   | 313       | 6.26           |
| 100  | 602       | 6.02           |
| 1000 | 6917      | 6.917          |

可以看到，随着消息数量增加，平均每个消息时间开销先急剧减小，之后不怎么变化，甚至略微增加。背后的原因是固定开销摊销 (Fixed Overhead Amortization)。DPU 与 Host 通信存在显著的固定开销摊销效应。单条消息通信需要承担完整的连接建立、协议初始化等固定成本 (57 s)，而批量通信将这些固定开销分摊到多条消息中，使平均单条消息延迟降至 6-7 s。这表明批量处理是提升 DPU-Host 通信性能的有效策略。这样的现象也给我们一些设计 DPU 通信的提示：

- 批量处理：尽量积攒消息后批量发送
- 连接复用：保持长连接，避免频繁建立
- 缓冲设计：合理的缓冲区大小平衡延迟和吞吐量

1.7 Buffer 大小对性能的影响分析

为了探究不同 Buffer 大小对数据传输性能的影响，我们设计了两组实验。第一组实验发送 100 条 1024 字节的消息，模拟大数据量传输场景。第二组实验发送 10 条 256 字节的消息，模拟小数据量、高频率的传输场景。实验分别测试了 Buffer 大小为 64、128、512 和 1024 字节时的性能指标，包括发送时间、接收时间和总传输时间。

1.7.1 大数据量传输性能（100 条 1024 字节消息）

表 3 展示了在发送 100 条 1024 字节消息时，不同 Buffer 大小下的性能数据。

表 3: 发送 100 条 1024 字节消息性能对比

| Buffer 大小 | 发送时间 ( $\mu s$ ) | 接收时间 ( $\mu s$ ) | 总传输时间 ( $\mu s$ ) |
|-----------|------------------|------------------|-------------------|
| 64        | 3895.49          | 175.56           | 3895.49           |
| 128       | 3928.41          | 202.16           | 3928.41           |
| 512       | 4157.32          | 197.12           | 4157.32           |
| 1024      | 3181.09          | 185.36           | 3181.09           |

从表 3 可以看出：

- **发送与总传输时间：**当 Buffer 大小设置为 1024 字节时，发送时间和总传输时间均达到最短，约为 3181 $\mu s$ ，性能表现最佳。这表明当 Buffer 大小与消息大小匹配时，可以有效减少因数据分片和重组带来的额外开销。相比之下，Buffer 为 512 字节时性能最差，耗时超过 4150 $\mu s$ 。
- **接收时间：**接收时间的差异相对较小，均在 175 $\mu s$  到 203 $\mu s$  的范围内。这说明在当前测试场景下，接收端的处理逻辑对 Buffer 大小的变化不敏感。

### 1.7.2 小数据量传输性能（10 条 256 字节消息）

表 4 展示了在发送 10 条 256 字节消息时，不同 Buffer 大小下的性能数据。

表 4: 发送 10 条 256 字节消息性能对比

| Buffer 大小 | 发送时间 ( $\mu s$ ) | 接收时间 ( $\mu s$ ) | 总传输时间 ( $\mu s$ ) |
|-----------|------------------|------------------|-------------------|
| 64        | 118.94           | 131.10           | 504.83            |
| 128       | 110.05           | 61.33            | 375.83            |
| 512       | 145.25           | 158.89           | 606.14            |
| 1024      | 443.85           | 253.78           | 556.45            |

从表 4 可以看出：

- **发送与总传输时间：**当 Buffer 大小设置为 128 字节时，总传输时间最短（约  $376\mu s$ ），获得了最佳性能。有趣的是，当 Buffer 大小远大于消息尺寸时（例如 1024 字节），发送时间急剧增加至  $443\mu s$ ，导致整体性能下降。这可能是由于过大的 Buffer 在处理小数据包时引入了不必要的内存操作延迟。
- **接收时间：**同样地，128 字节的 Buffer 在接收端也表现出最优性能，接收时间仅为  $61\mu s$ 。

### 1.7.3 总结

综合以上两组实验，可以得出以下结论：

1. **Buffer 大小与数据负载的匹配至关重要：**传输性能并非随着 Buffer 大小的增加而单调提升。最佳性能通常在 Buffer 大小与单次传输的数据量相匹配或略大时出现。
2. **大数据量场景：**对于大数据量的传输（如 1024 字节），较大的 Buffer（1024 字节）能够显著减少数据包处理的次数，从而降低系统开销，提升传输效率。
3. **小数据量场景：**对于小数据量、高频率的传输（如 256 字节），选择一个适中的、较小的 Buffer（如 128 字节）更为高效。过大的 Buffer 反而会因为资源分配和管理开销导致性能下降。

## 2 DPA All-to-All 通信

传统 CPU 中，MPI 的 All-to-All 通信需要 CPU 负责所有数据拷贝与交换，导致计算与通信难以并行，从而成为性能瓶颈。DOCA DPA（Data Path Accelerator）可将此类通信卸载至 DPU，加速多进程集体通信，并释放 CPU 资源。

### 2.1 编译与运行

使用 Meson 和 Ninja 编译，只启用 DPA All-to-All：

```
meson /tmp/try/dpa_a2a -Denable_all_applications=false -Denable_dpa_all_to_all=true
ninja -C /tmp/try/dpa_a2a
```

使用 MPI 启动并设置进程数与消息长度：

```
mpirun -np 4 /tmp/try/dpa_a2a/dpa_all_to_all/doca_dpa_all_to_all -m 32 -d "mlx5_0"
```

根据 NVIDIA 指南，该程序会将通信操作卸载至 DPA，减少 CPU 干预。

复现结果如图：



```

ubuntu@localhost:/tmp/build/dpa_all_to_all$ mpirun -np 4 /tmp/build/dpa_all_to_all/doca_dpa_all_to_
[13:54:00:848084][3407659][DOCA][INF][dpa_all_to_all_core.c:1749][dpa_a2a] Number of processes = 4,
-----send buffs-----
Rank 0 | 9923  9245  755   8690  3668  5887  91   5825 |
Rank 1 | 8443  835   9862  1115  2814  3320  6288  2955 |
Rank 2 | 3697  4560  3877  578   5535  2346  6483  5165 |
Rank 3 | 597   1348  774   4178  4379  8406  2055  648  |
-----recv buffs-----
Rank 0 | 9923  9245  8443  835   3697  4560  597   1348 |
Rank 1 | 755   8690  9862  1115  3877  578   774   4178 |
Rank 2 | 3668  5887  2814  3320  5535  2346  4379  8406 |
Rank 3 | 91    5825  6288  2955  6483  5165  2055  648  |
ubuntu@localhost:/tmp/build/dpa_all_to_all$

```

图 14: All to all 复现结果

## 2.2 源代码修改与性能测量

我们在 host 端代码中插入时间戳，进行以下实验：

表 5: 消息长度对比（4 进程）»

| 消息长度（字节） | 通信时间（秒）  |
|----------|----------|
| 16       | 1.521223 |
| 32       | 1.525354 |
| ...      | ...      |
| 4096     | 1.521331 |

表 6: 线程数量对比（消息长度 = 64）»

| 线程数 | 通信时间（秒）  |
|-----|----------|
| 2   | 0.919402 |
| 4   | 1.522374 |
| 8   | 2.813593 |
| 16  | 5.647253 |

分析：线程数增加导致时间显著上升，而通信时间稳定（约 1.52s），与消息长度关联不大。结合以上的结果发现 all-to-all 通信时间主要花在将大的消息缓冲区拆分成多个小块并发送 RDMA 请求，以及持续检查这些请求是否完成（即“发布 RDMA 操作 + 轮询”。）换句话说，就是在把数据切成片、发出去、再不断地查看“有没有发完”这几步上耗费了绝大部分时间。

在实践中线程数应控制在 2-4，以获得最佳性能。但是与传统 CPU MPI 对比之下，进程数从 4 增至 16 时，通信时间近线性增长，DPU DPA 的效果仍然显著。

## 2.3 Allreduce 应用探索

为了探究 DPU 对其他的 MPI 应用的优化效果，通过参考官方文档：[NVIDIA DOCA All-Reduce 应用指南](#)，小组实现了对 AllReduce 功能的复现。

### 2.3.1 原理说明

Allreduce 是在多个进程间执行聚合操作的通信模式，例如多个设备分别计算出自己的部分张量的梯度，然后通过 AllReduce 统一发送、累加，最终使每个设备得到全局的“同一个”梯度结果。DOCA 支持两种模式：

- **Offloaded client**：Host 端提交归约请求，DPU 上 daemon 负责完成操作；
- **Non-offloaded client**：Host 自行执行所有通信与计算。

架构上，offload 能实现计算与通信重叠，提高整体效率。

### 2.3.2 环境配置与运行方式

在测试初期遇到了一些问题：如未设置 ipv4 地址，运行时段错误。经过一系列的调试可以通过以下命令实现进程间的通信。构建命令：

```
meson /tmp/try/allreduce -Denable_all_applications=false -Denable_allreduce=true
ninja -C /tmp/try/allreduce
```

运行示例：

```
export UCX_NET_DEVICES=mlx5_2:1 # 设置 BF 设备
# 守护进程端 （运行在 DPU 上）
UCX_LOG_LEVEL=info ./doca_allreduce -r daemon -t 35001 -c 2 -a 192.168.1.100:35001 \
    -s 65535 -o sum -d float -i 16 -b 128
# 客户端
./doca_allreduce -r client -m non-offloaded -t 36001 -a 192.168.1.100:35001 \
    -s 65535 -i 16 -b 128 -o sum -d float
```

### 2.3.3 初步性能对比

官方文档中，Offload 模式的通信时间显著低于非 offload，且可实现约 20%–30% 的通信与计算重叠。在 Offload 模式下，通信与计算可以重叠进行：Host 在提交 AllReduce 请求后，可以立即返回继续计算，而 DPU 在后台完成通信。实践中，我们测得数据为：

- Offload 模式：通信耗时约为 0.8s，计算耗时约 1.2s，通信与计算实际重叠，总体缩短约 20%；
- Non-offload 模式：通信耗时约为 1.0s，计算耗时 1.2s，总耗时 = 2.2s，无重叠；

在数据并行训练场景中，模型前向与反向计算往往进行到一半就需要同步梯度。若此时通信阻塞主机计算，则会导致 GPU/CPU 利用率下降。使用 Offload 模式后，主机可持续进行计算并等待 DPU 完成通信。由此可提升硬件利用率和整体训练吞吐，从而在训练效率与资源利用两个方面带来显著优势。