Optimization of RDMA communication on an Infiniband network

Presentation no .1

Stamoulis Spyros 03775

Professors:

- Christos Antonopoulos
- Athanasios Fevgas

Initial steps

- Studied the samples
  - rdma_send/rdma_receive
  - rdma_write,rdma_read
- After I understood the functionality of the code I tried to generalize the send receive sample to send many messages.

Version 1

- As a first change I changed the rdma_send/rdma_receive from a single message program to send/receive 1000 messages and benchmarking the time.

Goal

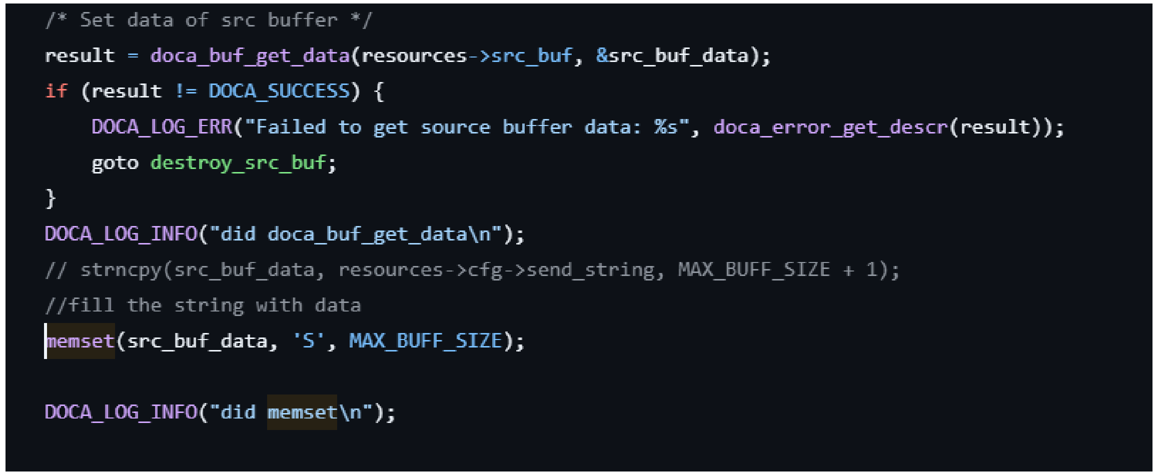
- **Original (Before):** The program was designed to:
  - Connect to the receiver.
  - Prepare *one* buffer.
  - Submit *one* send task.
  - Clean up and exit.
- **New Goal (After):** The benchmark needs to:
  - Connect to the receiver.
  - Start a timer.
  - Send packets.
  - Stop the timer.
  - Calculate and report the throughput (Gbps).

Send sample

Filling the *Entire* Buffer

- To measure throughput, we must send the full buffer size, not just a small string.
- I replaced strncpy (which copies a short string) with memset.
- This ensures we are benchmarking the full MAX_BUFF_SIZE (e.g., 1MB) on every send.

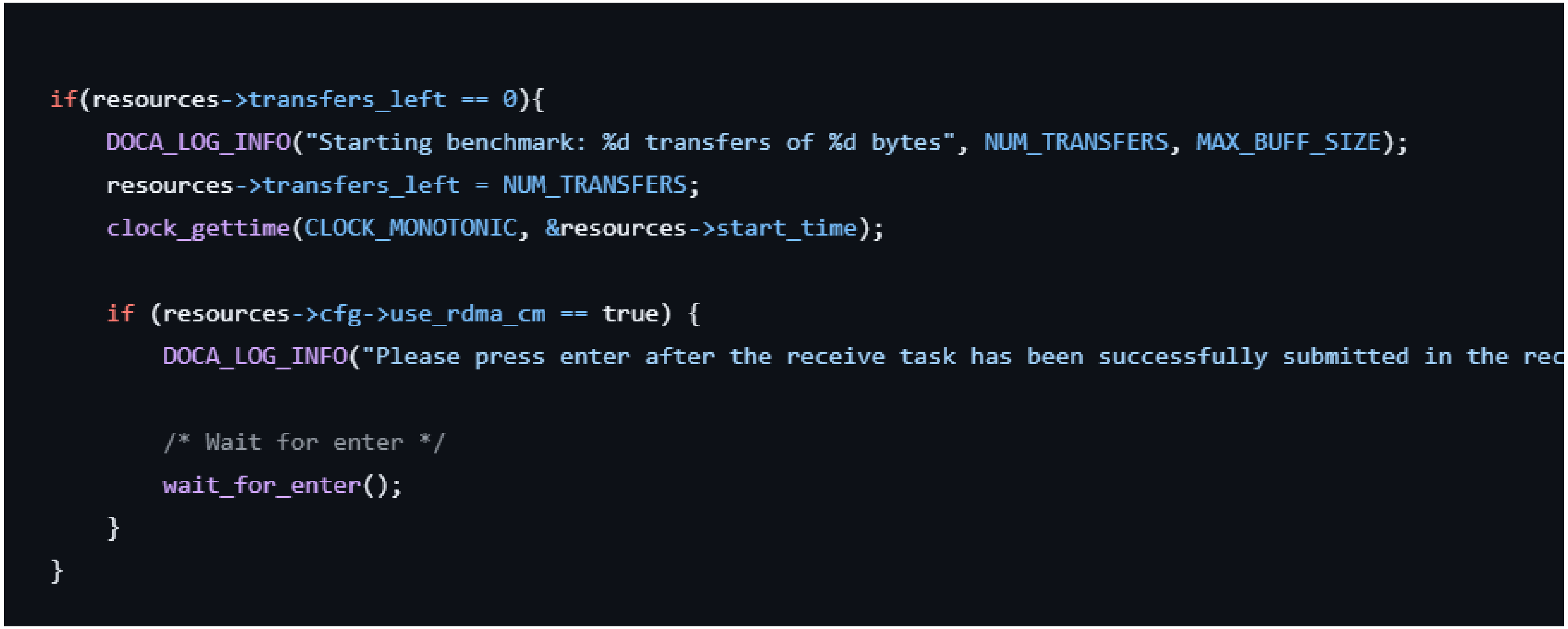
The image on the right is from the func prepare and submit task:



```
/* Set data of src buffer */
result = doca_buf_get_data(resources->src_buf, &src_buf_data);
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to get source buffer data: %s", doca_error_get_descr(result));
    goto destroy_src_buf;
}
DOCA_LOG_INFO("did doca_buf_get_data\n");
// strncpy(src_buf_data, resources->cfg->send_string, MAX_BUFF_SIZE + 1);
//fill the string with data
memset(src_buf_data, 'S', MAX_BUFF_SIZE);

DOCA_LOG_INFO("did memset\n");
```

Starting the Benchmark

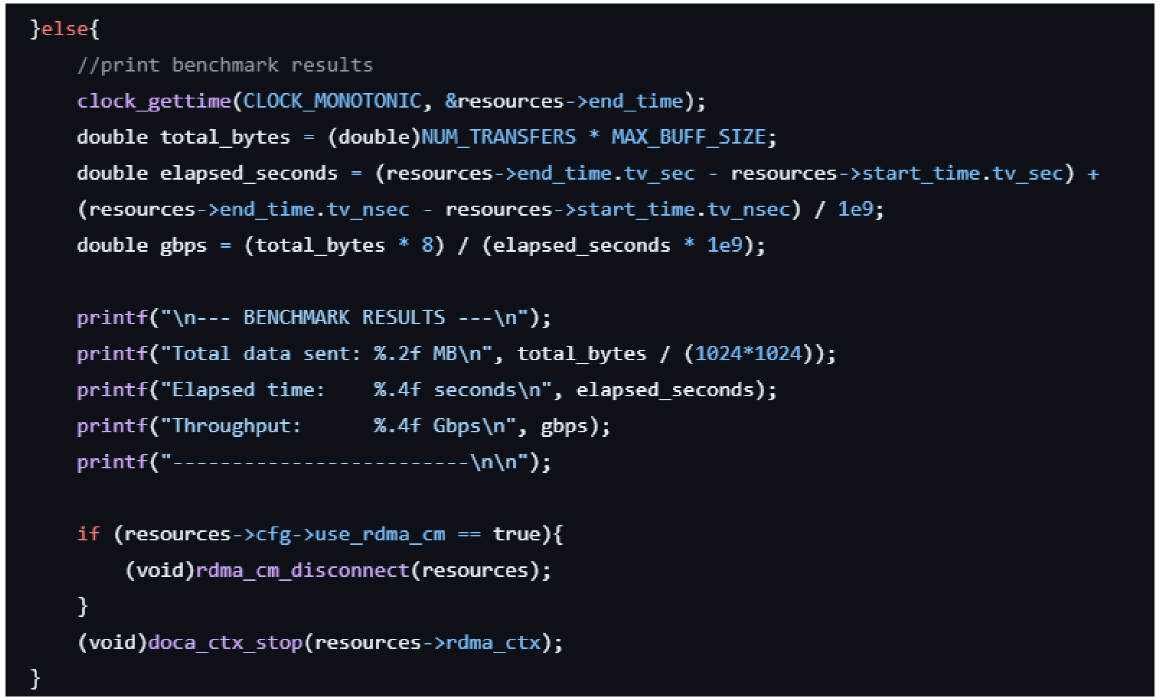
- Added logic to rdma_send_prepare_and_submit_task to start the benchmark.
- It uses a new counter, transfers_left, to track the state.
- On the very first call (transfers_left == 0), it starts the high-resolution timer.



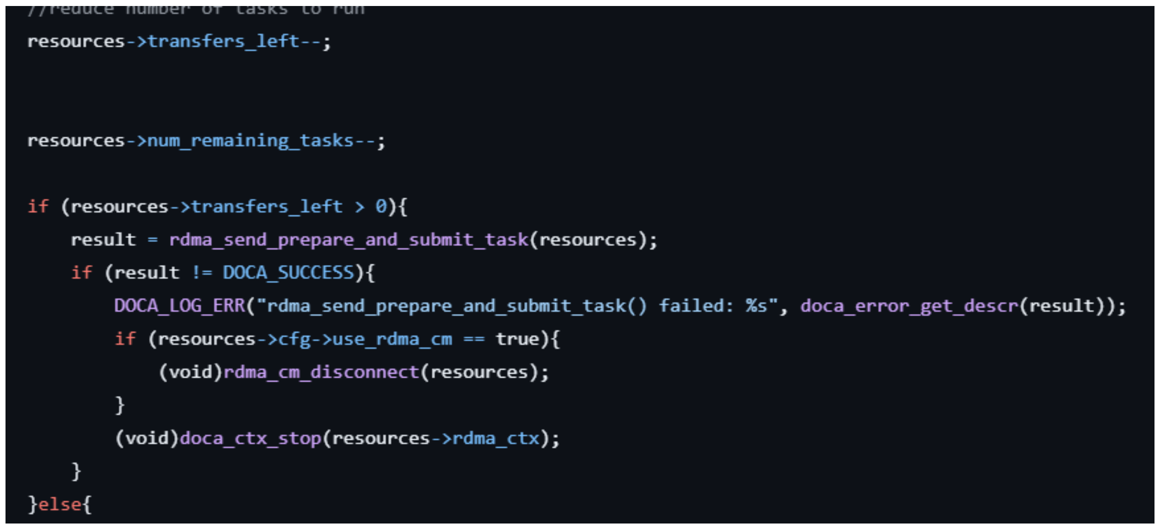
```
if(resources->transfers_left == 0){
    DOCA_LOG_INFO("Starting benchmark: %d transfers of %d bytes", NUM_TRANSFERS, MAX_BUFF_SIZE);
    resources->transfers_left = NUM_TRANSFERS;
    clock_gettime(CLOCK_MONOTONIC, &resources->start_time);

    if (resources->cfg->use_rdma_cm == true) {
        DOCA_LOG_INFO("Please press enter after the receive task has been successfully submitted in the rec

        /* Wait for enter */
        wait_for_enter();
    }
}
```

The Benchmark Logic

- The completion callback (rdma_send_completed_callback) became the heart of the new version.
- When a send completes, the callback does the following things:
    1. Decrements the transfers_left counter.
    2. If transfers_left > 0, it immediately calls rdma_send_prepare_and_submit_task again.
    3. Ends the benchmark if there are no more tasks to do.



```
}else{
    //print benchmark results
    clock_gettime(CLOCK_MONOTONIC, &resources->end_time);
    double total_bytes = (double)NUM_TRANSFERS * MAX_BUFF_SIZE;
    double elapsed_seconds = (resources->end_time.tv_sec - resources->start_time.tv_sec) +
    (resources->end_time.tv_nsec - resources->start_time.tv_nsec) / 1e9;
    double gbps = (total_bytes * 8) / (elapsed_seconds * 1e9);

    printf("\n--- BENCHMARK RESULTS ---\n");
    printf("Total data sent: %.2f MB\n", total_bytes / (1024*1024));
    printf("Elapsed time:    %.4f seconds\n", elapsed_seconds);
    printf("Throughput:      %.4f Gbps\n", gbps);
    printf("-------------------------\n\n");

    if (resources->cfg->use_rdma_cm == true){
        (void)rdma_cm_disconnect(resources);
    }
    (void)doca_ctx_stop(resources->rdma_ctx);
}
```



```
//reduce number of tasks to run
resources->transfers_left--;


resources->num_remaining_tasks--;

if (resources->transfers_left > 0){
    result = rdma_send_prepare_and_submit_task(resources);
    if (result != DOCA_SUCCESS){
        DOCA_LOG_ERR("rdma_send_prepare_and_submit_task() failed: %s", doca_error_get_descr(result));
        if (resources->cfg->use_rdma_cm == true){
            (void)rdma_cm_disconnect(resources);
        }
        (void)doca_ctx_stop(resources->rdma_ctx);
    }
}else{
```

Receive sample

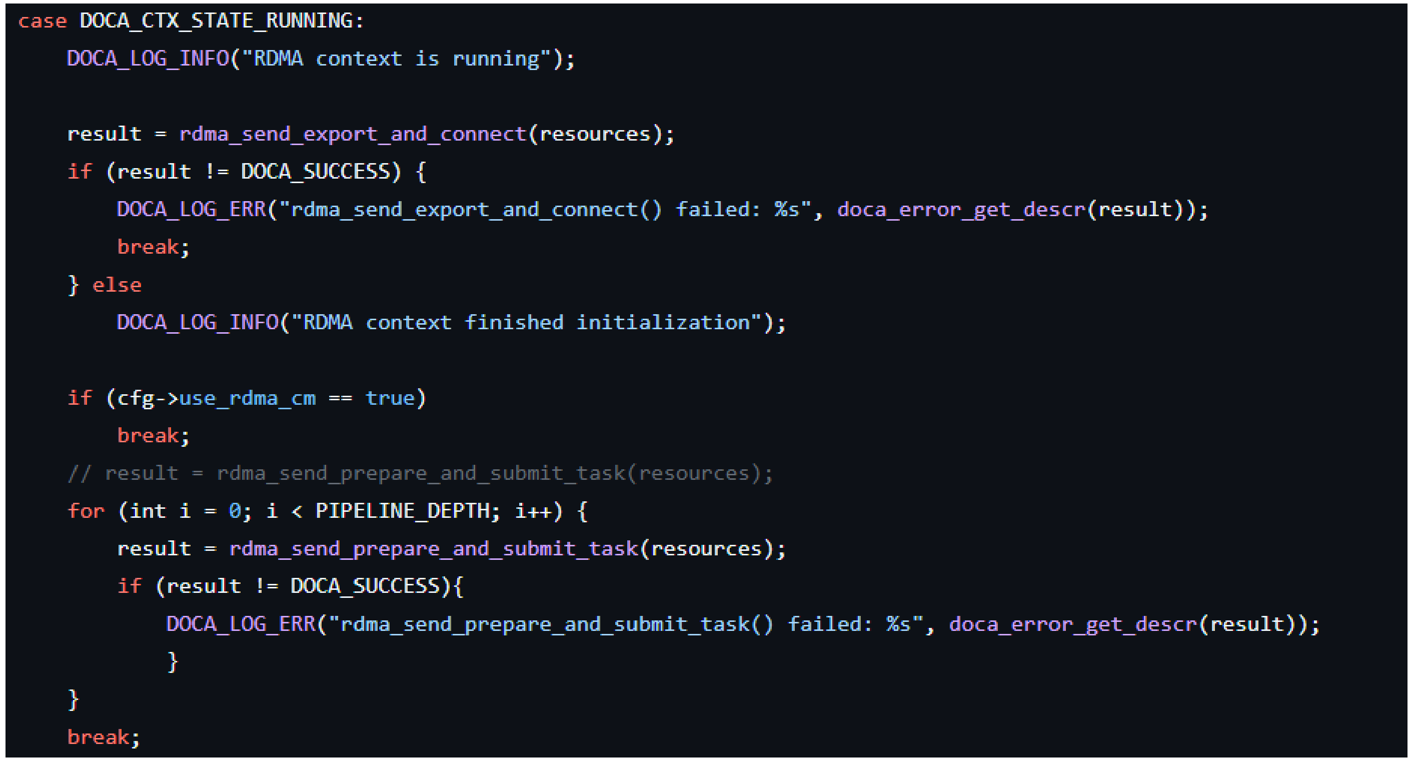
- On the receive side I did almost the same changes to be able to receive NUM_TRANSFERS number of messages

Version 2

- Here I tried to implement a pipeline of the tasks to improve the throughput and I experimented with different pipe sizes
- I only changed the send file

In the rdma_send_state_change_callback

I submit PIPELINE DEPTH tasks instead of just one in the beggining



```
case DOCA_CTX_STATE_RUNNING:
    DOCA_LOG_INFO("RDMA context is running");

    result = rdma_send_export_and_connect(resources);
    if (result != DOCA_SUCCESS) {
        DOCA_LOG_ERR("rdma_send_export_and_connect() failed: %s", doca_error_get_descr(result));
        break;
    } else
        DOCA_LOG_INFO("RDMA context finished initialization");

    if (cfg->use_rdma_cm == true)
        break;
    // result = rdma_send_prepare_and_submit_task(resources);
    for (int i = 0; i < PIPELINE_DEPTH; i++) {
        result = rdma_send_prepare_and_submit_task(resources);
        if (result != DOCA_SUCCESS){
            DOCA_LOG_ERR("rdma_send_prepare_and_submit_task() failed: %s", doca_error_get_descr(result));
            }
    }
    break;
```

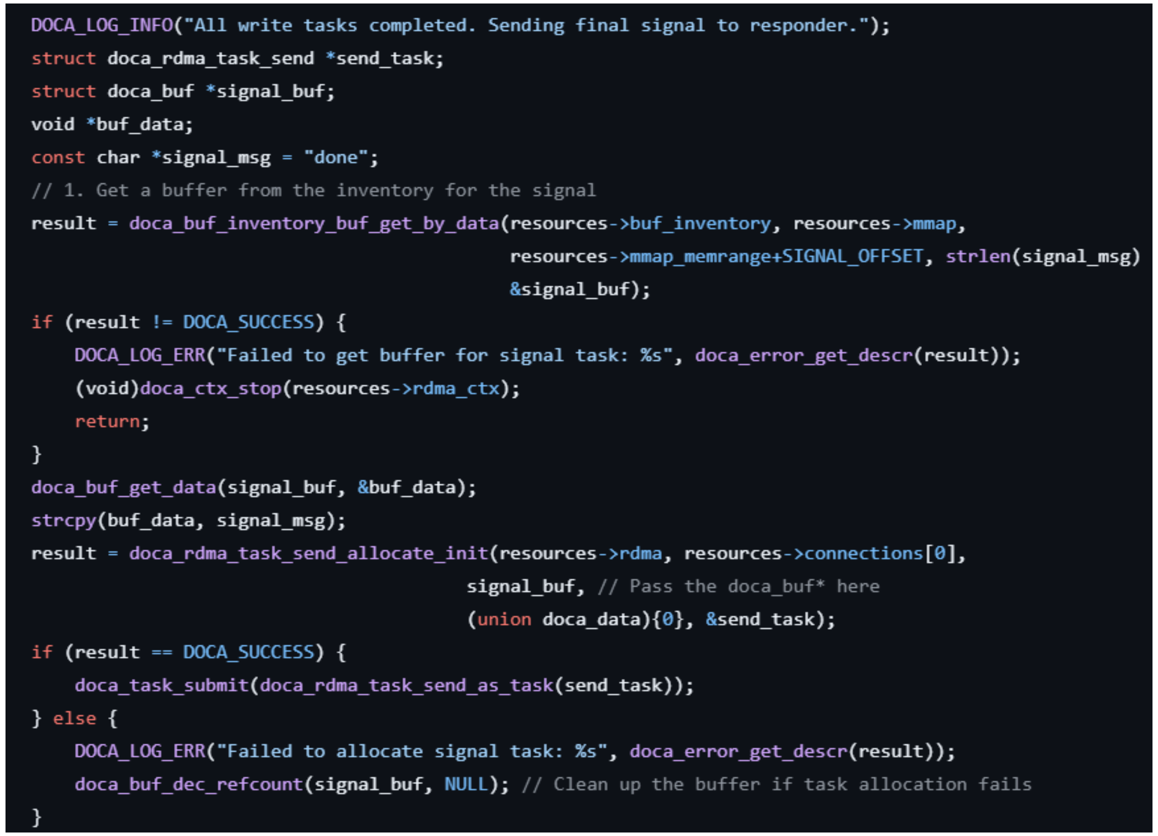
Version 3

- I moved to the write sample and I did the same changes in the requester sample as the ones I did at version 1 on the send sample in order to make possible the multiple write tasks.
- This sample was waiting for the user to press enter to finish so I added a send task to let the receiver (write_responder) that the writing has finished. Before the send task, the sender (write_requester) stops the benchmark and prints the results.
- Finally when the receiver gets the final send task it stops the context.

Requester (client)

- Added send task when the benchmark is finished

- Also added standard callbacks to handle the ending of the send task.

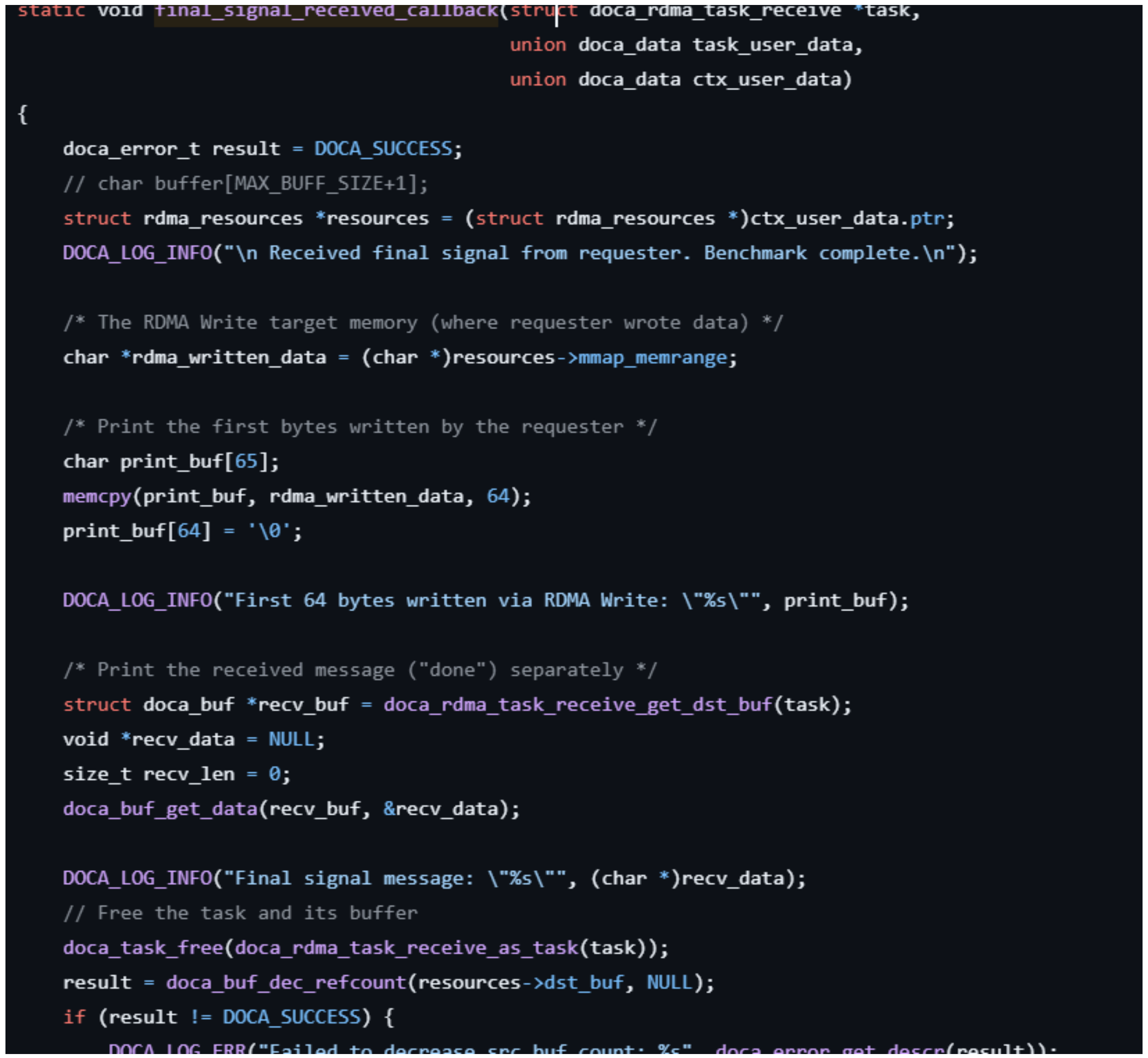


```c
DOCA_LOG_INFO("All write tasks completed. Sending final signal to responder.");
struct doca_rdma_task_send *send_task;
struct doca_buf *signal_buf;
void *buf_data;
const char *signal_msg = "done";
// 1. Get a buffer from the inventory for the signal
result = doca_buf_inventory_buf_get_by_data(resources->buf_inventory, resources->mmap,
                                            resources->mmap_memrange+SIGNAL_OFFSET, strlen(signal_msg)
                                            &signal_buf);
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to get buffer for signal task: %s", doca_error_get_descr(result));
    (void)doca_ctx_stop(resources->rdma_ctx);
    return;
}
doca_buf_get_data(signal_buf, &buf_data);
strcpy(buf_data, signal_msg);
result = doca_rdma_task_send_allocate_init(resources->rdma, resources->connections[0],
                                          signal_buf, // Pass the doca_buf* here
                                          (union doca_data){0}, &send_task);
if (result == DOCA_SUCCESS) {
    doca_task_submit(doca_rdma_task_send_as_task(send_task));
} else {
    DOCA_LOG_ERR("Failed to allocate signal task: %s", doca_error_get_descr(result));
    doca_buf_dec_refcount(signal_buf, NULL); // Clean up the buffer if task allocation fails
}
```

Responder (server)

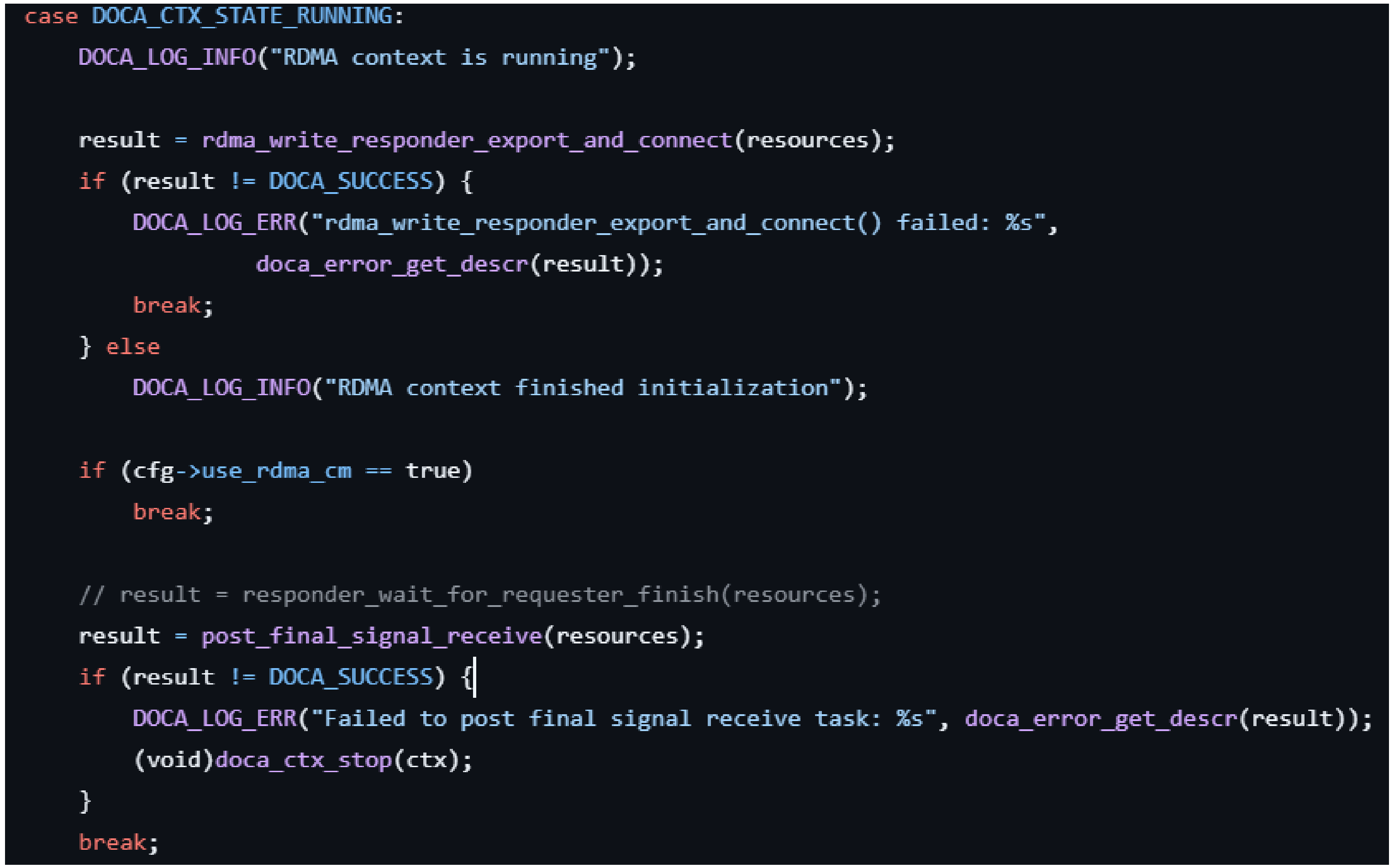
- Added final_signal_received_callback handles when the receive task is completed and checks 64 bytes in the memory to make sure the write operation was correct.

- I also added a standard error callback func that I copied from the receive sample
- And in similar fashion a post receive task func



```
static void final_signal_received_callback(struct doca_rdma_task_receive *task,
                                           union doca_data task_user_data,
                                           union doca_data ctx_user_data)
{
    doca_error_t result = DOCA_SUCCESS;
    // char buffer[MAX_BUFF_SIZE+1];
    struct rdma_resources *resources = (struct rdma_resources *)ctx_user_data.ptr;
    DOCA_LOG_INFO("\n Received final signal from requester. Benchmark complete.\n");

    /* The RDMA Write target memory (where requester wrote data) */
    char *rdma_written_data = (char *)resources->mmap_memrange;

    /* Print the first bytes written by the requester */
    char print_buf[65];
    memcpy(print_buf, rdma_written_data, 64);
    print_buf[64] = '\0';

    DOCA_LOG_INFO("First 64 bytes written via RDMA Write: \"%s\"", print_buf);

    /* Print the received message ("done") separately */
    struct doca_buf *recv_buf = doca_rdma_task_receive_get_dst_buf(task);
    void *recv_data = NULL;
    size_t recv_len = 0;
    doca_buf_get_data(recv_buf, &recv_data);

    DOCA_LOG_INFO("Final signal message: \"%s\"", (char *)recv_data);
    // Free the task and its buffer
    doca_task_free(doca_rdma_task_receive_as_task(task));
    result = doca_buf_dec_refcount(resources->dst_buf, NULL);
    if (result != DOCA_SUCCESS) {
        DOCA_LOG_ERR("Failed to decrease src buf count: %s", doca_error_get_descr(result));
```

Also in the state change func I changed the function called. So instead of actively waiting it waits for a receive task to finish



```
case DOCA_CTX_STATE_RUNNING:
    DOCA_LOG_INFO("RDMA context is running");

    result = rdma_write_responder_export_and_connect(resources);
    if (result != DOCA_SUCCESS) {
        DOCA_LOG_ERR("rdma_write_responder_export_and_connect() failed: %s",
                 doca_error_get_descr(result));
        break;
    } else
        DOCA_LOG_INFO("RDMA context finished initialization");

    if (cfg->use_rdma_cm == true)
        break;

    // result = responder_wait_for_requester_finish(resources);
    result = post_final_signal_receive(resources);
    if (result != DOCA_SUCCESS) {
        DOCA_LOG_ERR("Failed to post final signal receive task: %s", doca_error_get_descr(result));
        (void)doca_ctx_stop(ctx);
    }
    break;
```

Times

- Note:
  - For all tests total data sent= 62.50 MB.
  - For the previous versions, the throughput is negligible (near zero)
- Times:
  - Throughput: 5.1881 Gbps
  - Throughput: 2.8214 Gbps
  - Throughput: 5.1820 Gbps
  - Throughput: 3.8275 Gbps
  - Throughput: 5.1470 Gbps
- Average = 4.433200
- Standard deviation = 1.072947

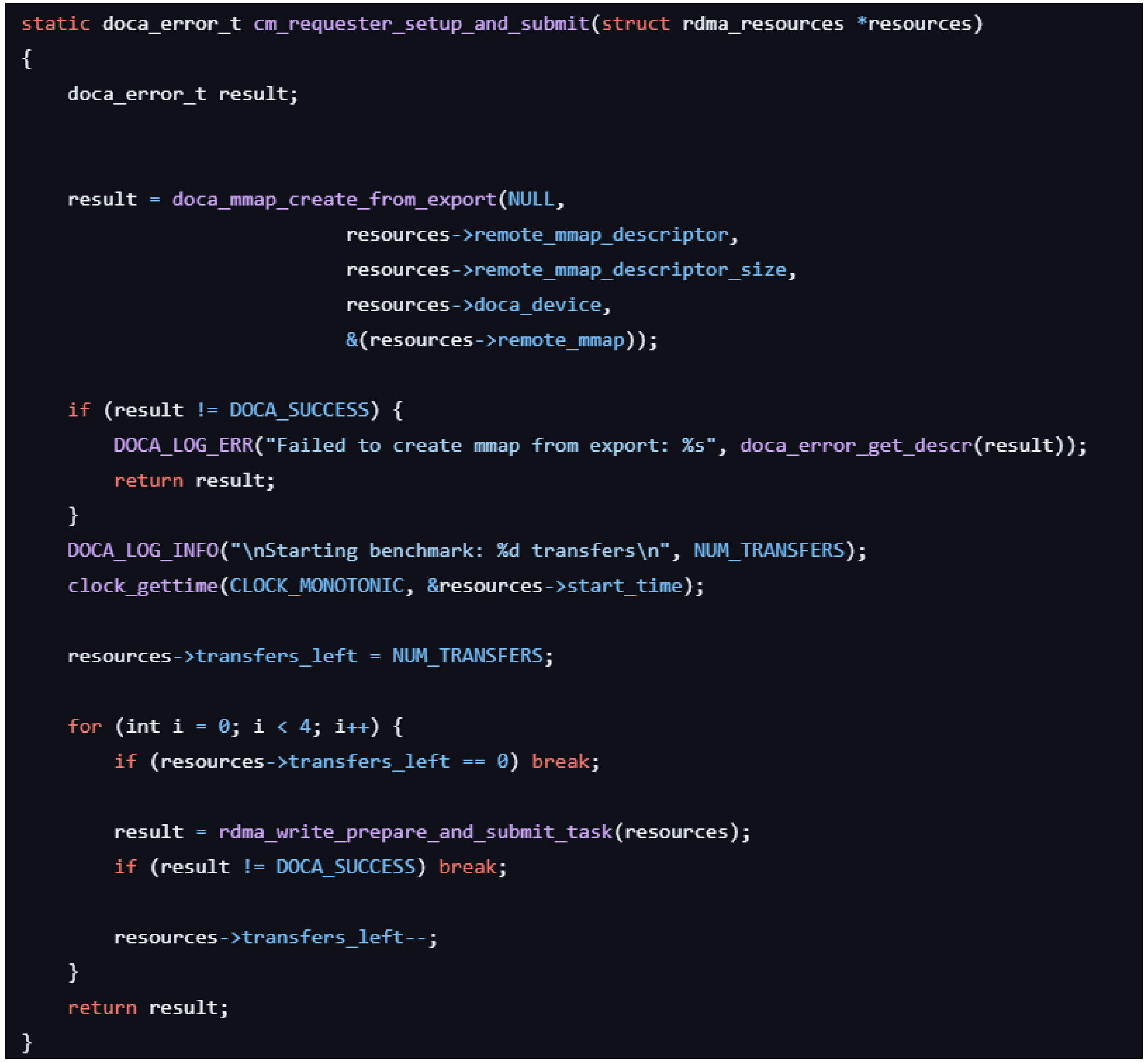
Version 4 (Also fixed a few issues)

- I added pipeline and I exported only once the mmap instead of doing it in every task submission.

- Each message overwrites the previous one
- Issue with the freeing of the buffers
- When the 1000th task is submitted the benchmark ends. So we don’t wait for it to finish

Requester

I added a function to fill the pipeline in the beginning and export the mmap



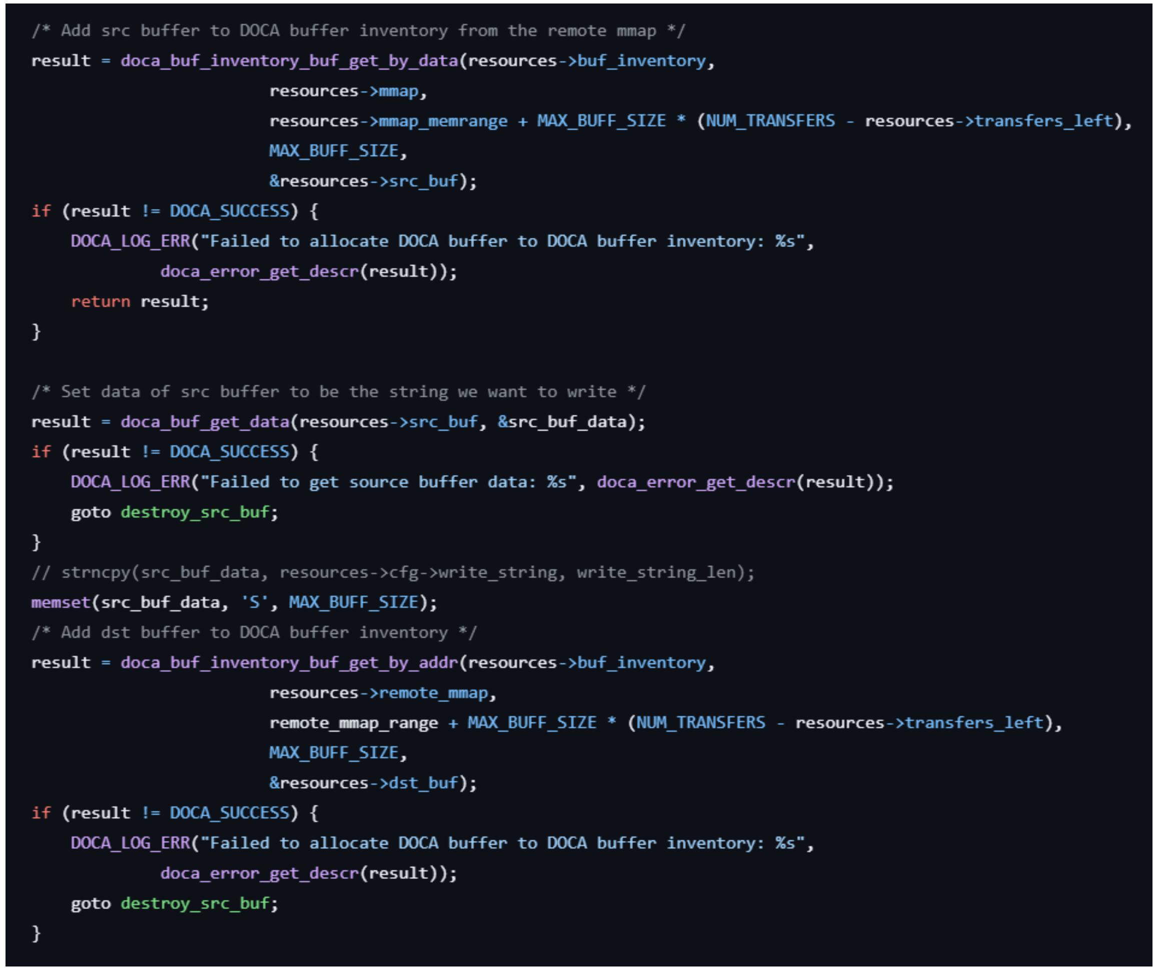
```c
static doca_error_t cm_requester_setup_and_submit(struct rdma_resources *resources)
{
    doca_error_t result;


    result = doca_mmap_create_from_export(NULL,
                                          resources->remote_mmap_descriptor,
                                          resources->remote_mmap_descriptor_size,
                                          resources->doca_device,
                                          &(resources->remote_mmap));

    if (result != DOCA_SUCCESS) {
        DOCA_LOG_ERR("Failed to create mmap from export: %s", doca_error_get_descr(result));
        return result;
    }
    DOCA_LOG_INFO("\nStarting benchmark: %d transfers\n", NUM_TRANSFERS);
    clock_gettime(CLOCK_MONOTONIC, &resources->start_time);

    resources->transfers_left = NUM_TRANSFERS;

    for (int i = 0; i < 4; i++) {
        if (resources->transfers_left == 0) break;

        result = rdma_write_prepare_and_submit_task(resources);
        if (result != DOCA_SUCCESS) break;

        resources->transfers_left--;
    }
    return result;
}
```

Buffer offset

- While buffers were writing in the same place in the previous version, now with the pipeline they cannot do that as one would overwrite the other. So, I introduced offset to the memory address each one was writing at.

- The offset I introduced was:

MAX_BUFF_SIZE * (NUM_TRANSFERS - resources->transfers_left)



```
/* Add src buffer to DOCA buffer inventory from the remote mmap */
result = doca_buf_inventory_buf_get_by_data(resources->buf_inventory,
                        resources->mmap,
                        resources->mmap_memrange + MAX_BUFF_SIZE * (NUM_TRANSFERS - resources->transfers_left),
                        MAX_BUFF_SIZE,
                        &resources->src_buf);
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to allocate DOCA buffer to DOCA buffer inventory: %s",
             doca_error_get_descr(result));
    return result;
}

/* Set data of src buffer to be the string we want to write */
result = doca_buf_get_data(resources->src_buf, &src_buf_data);
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to get source buffer data: %s", doca_error_get_descr(result));
    goto destroy_src_buf;
}
// strncpy(src_buf_data, resources->cfg->write_string, write_string_len);
memset(src_buf_data, 'S', MAX_BUFF_SIZE);
/* Add dst buffer to DOCA buffer inventory */
result = doca_buf_inventory_buf_get_by_addr(resources->buf_inventory,
                        resources->remote_mmap,
                        remote_mmap_range + MAX_BUFF_SIZE * (NUM_TRANSFERS - resources->transfers_left),
                        MAX_BUFF_SIZE,
                        &resources->dst_buf);
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to allocate DOCA buffer to DOCA buffer inventory: %s",
             doca_error_get_descr(result));
    goto destroy_src_buf;
}
```

- Also fixed an error in the freeing of the buffers
- And fixed the premature ending of the benchmark by checking specifically if number of remaining tasks is zero.

Responder

- I added a random offset to the printed 64 bytes so I check if the memory is correctly written.
- Also in BOTH the files I introduced a var signal_offset that skips the written area in the memory in order for the send/receive buffer to use.

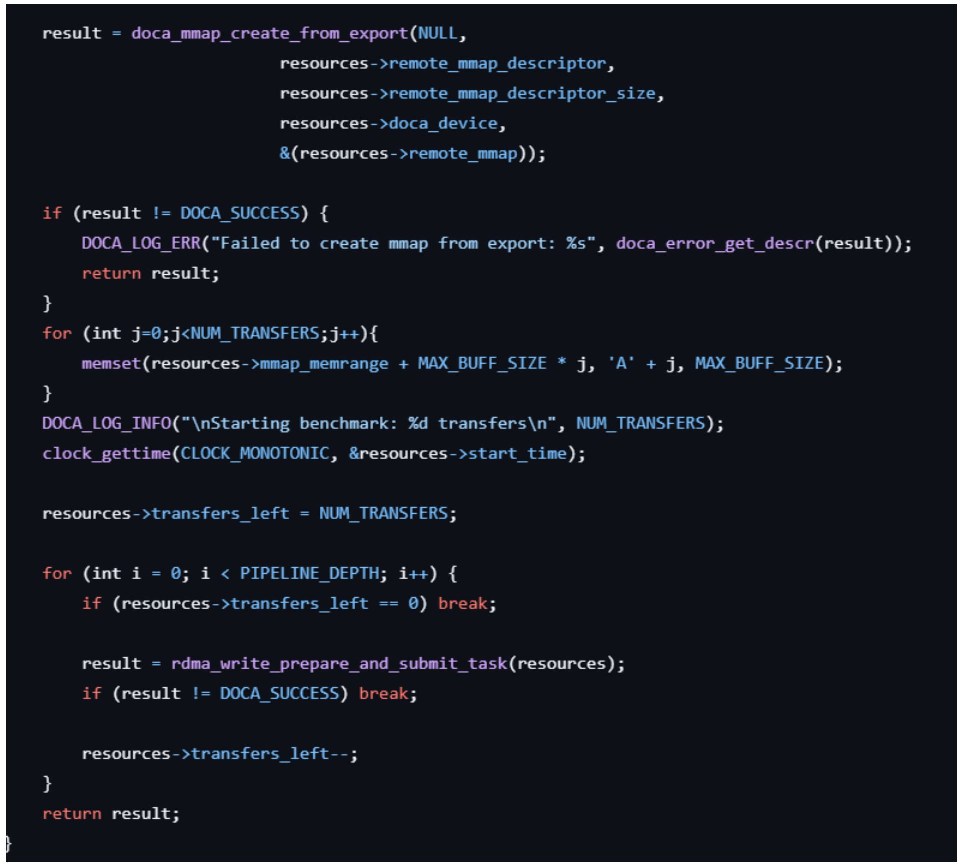
Times

- Times:
  - Throughput: 31.4731 Gbps
  - Throughput: 29.9341 Gbps
  - Throughput: 30.1986 Gbps
  - Throughput: 31.2671 Gbps
  - Throughput: 30.7210 Gbps
- Average = 30.718780
- Standard deviation = 0.662565

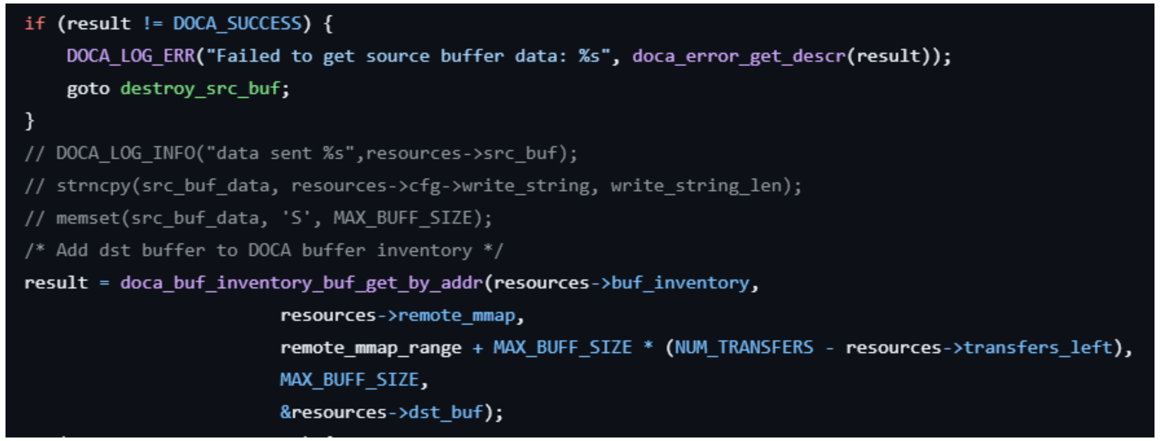
Version 5 (After last meeting)

Requester

- I filled the memory before hand in order to save time.
- Drastically increased message size. (All versions now have the same message size for comparison purposes, but I increased the size here to 64MB for the first time.)
- Also removed DOCA_LOG_INFO instructions from the submit write task func because they were wasting time and making excess noise.



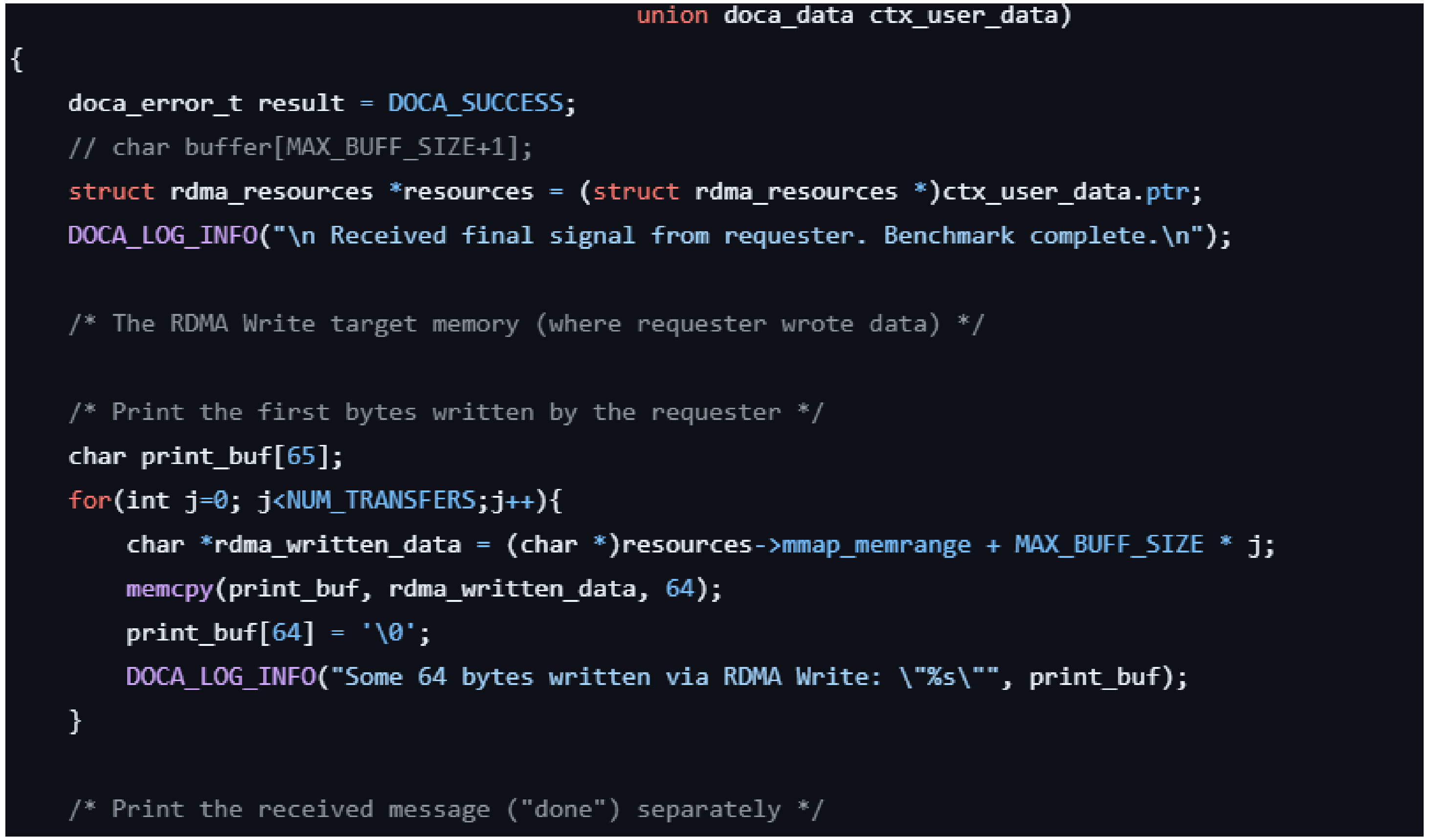
```
result = doca_mmap_create_from_export(NULL,
                        resources->remote_mmap_descriptor,
                        resources->remote_mmap_descriptor_size,
                        resources->doca_device,
                        &(resources->remote_mmap));

if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to create mmap from export: %s", doca_error_get_descr(result));
    return result;
}
for (int j=0;j<NUM_TRANSFERS;j++){
    memset(resources->mmap_memrange + MAX_BUFF_SIZE * j, 'A' + j, MAX_BUFF_SIZE);
}
DOCA_LOG_INFO("\nStarting benchmark: %d transfers\n", NUM_TRANSFERS);
clock_gettime(CLOCK_MONOTONIC, &resources->start_time);

resources->transfers_left = NUM_TRANSFERS;

for (int i = 0; i < PIPELINE_DEPTH; i++) {
    if (resources->transfers_left == 0) break;

    result = rdma_write_prepare_and_submit_task(resources);
    if (result != DOCA_SUCCESS) break;

    resources->transfers_left--;
}
return result;
}
```



```
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to get source buffer data: %s", doca_error_get_descr(result));
    goto destroy_src_buf;
}
// DOCA_LOG_INFO("data sent %s",resources->src_buf);
// strncpy(src_buf_data, resources->cfg->write_string, write_string_len);
// memset(src_buf_data, 'S', MAX_BUFF_SIZE);
/* Add dst buffer to DOCA buffer inventory */
result = doca_buf_inventory_buf_get_by_addr(resources->buf_inventory,
                        resources->remote_mmap,
                        remote_mmap_range + MAX_BUFF_SIZE * (NUM_TRANSFERS - resources->transfers_left),
                        MAX_BUFF_SIZE,
                        &resources->dst_buf);
```

Responder:

- I now print the first 64 bytes of each buffer.



```
                                                 union doca_data ctx_user_data)
{
    doca_error_t result = DOCA_SUCCESS;
    // char buffer[MAX_BUFF_SIZE+1];
    struct rdma_resources *resources = (struct rdma_resources *)ctx_user_data.ptr;
    DOCA_LOG_INFO("\n Received final signal from requester. Benchmark complete.\n");

    /* The RDMA Write target memory (where requester wrote data) */

    /* Print the first bytes written by the requester */
    char print_buf[65];
    for(int j=0; j<NUM_TRANSFERS;j++){
        char *rdma_written_data = (char *)resources->mmap_memrange + MAX_BUFF_SIZE * j;
        memcpy(print_buf, rdma_written_data, 64);
        print_buf[64] = '\0';
        DOCA_LOG_INFO("Some 64 bytes written via RDMA Write: \"%s\"", print_buf);
    }

    /* Print the received message ("done") separately */
```

Times

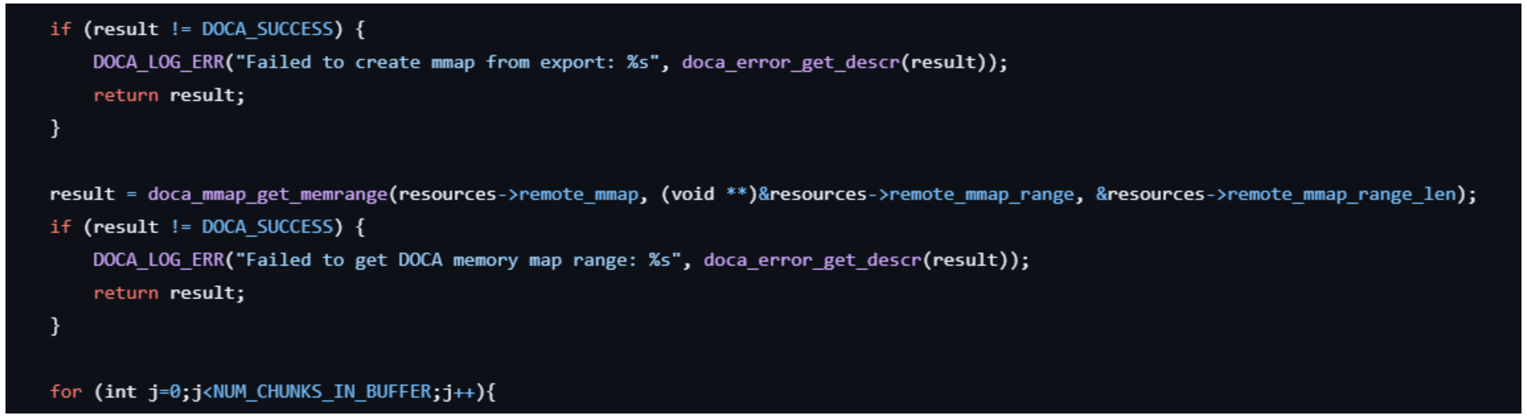
- Times:
  - Throughput: 64.0851 Gbps
  - Throughput: 48.3647 Gbps
  - Throughput: 46.9563 Gbps
  - Throughput: 46.3695 Gbps
  - Throughput: 43.2281 Gbps
- Average = 49.800740
- Standard deviation = 8.203185

Version 6:

- Requester:
  - Introduced a variable that shows the number of available buffers in memory (NUM_CHUNKS_IN_BUFFER)
  - And made possible to reuse the memory allowing to send more than MEMORY/MAM_BUFF_SIZE messages, by circling around.
  - I also fetch remote mmap only once.

- In responder fixed a small issue with freeing the buffer of the receive task.

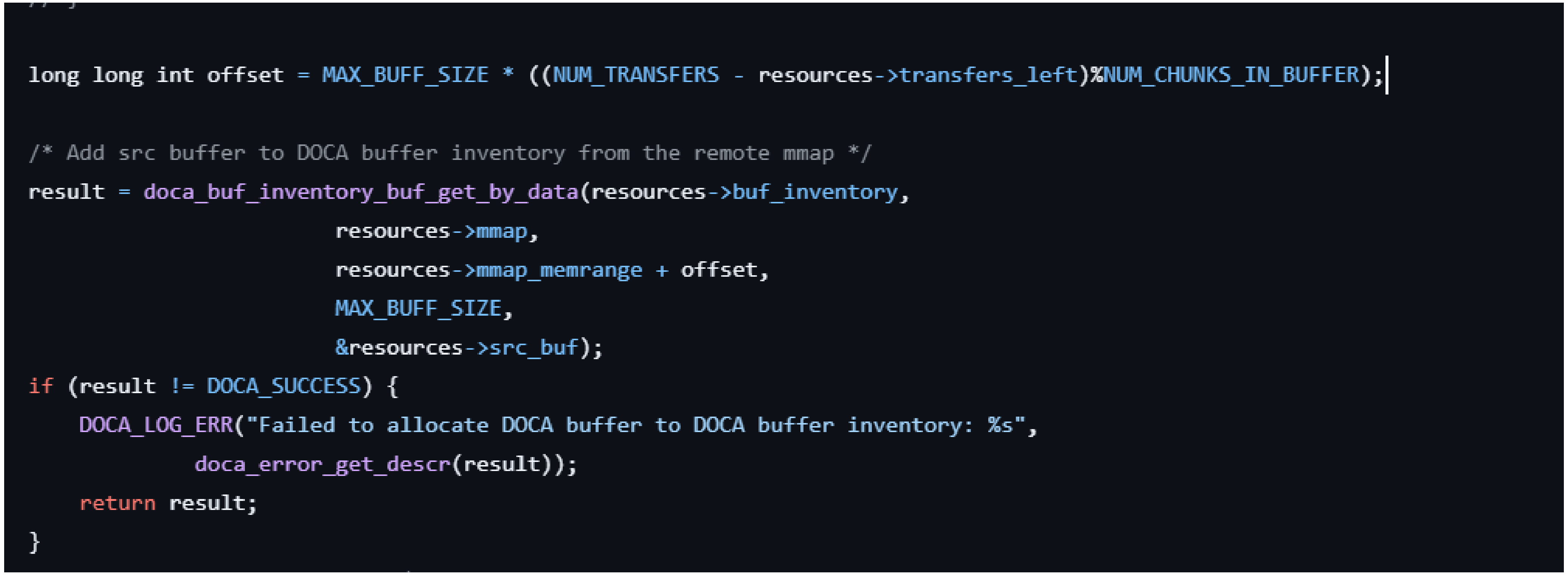
Getting remote mmap once



```
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to create mmap from export: %s", doca_error_get_descr(result));
    return result;
}

result = doca_mmap_get_memrange(resources->remote_mmap, (void **)&resources->remote_mmap_range, &resources->remote_mmap_range_len);
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to get DOCA memory map range: %s", doca_error_get_descr(result));
    return result;
}

for (int j=0;j<NUM_CHUNKS_IN_BUFFER;j++){
```

Circling logic

The image has the code for the local buffer. Its similar for the remote buffer



```
long long int offset = MAX_BUFF_SIZE * ((NUM_TRANSFERS - resources->transfers_left)%NUM_CHUNKS_IN_BUFFER);

/* Add src buffer to DOCA buffer inventory from the remote mmap */
result = doca_buf_inventory_buf_get_by_data(resources->buf_inventory,
                        resources->mmap,
                        resources->mmap_memrange + offset,
                        MAX_BUFF_SIZE,
                        &resources->src_buf);
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to allocate DOCA buffer to DOCA buffer inventory: %s",
            doca_error_get_descr(result));
    return result;
}
```

Times

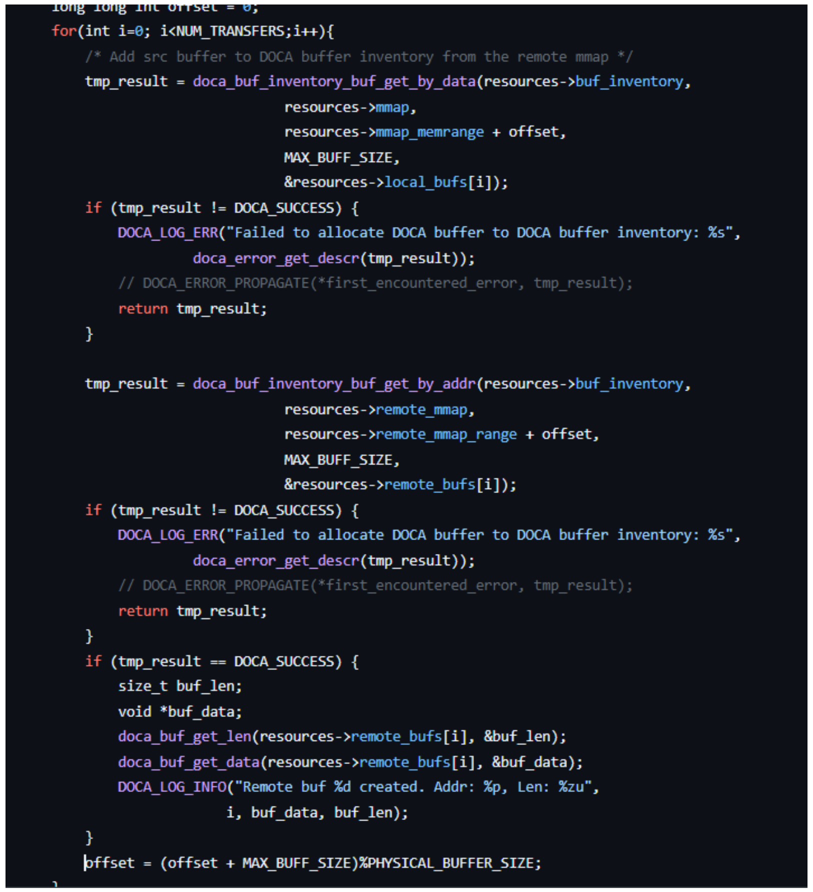
- Times:
  - Throughput: 46.5645 Gbps
  - Throughput: 47.4954 Gbps
  - Throughput: 43.8896 Gbps
  - Throughput: 64.3808 Gbps
  - Throughput: 48.4482 Gbps
- Average = 50.155700
- Standard deviation = 8.131835

Version 7

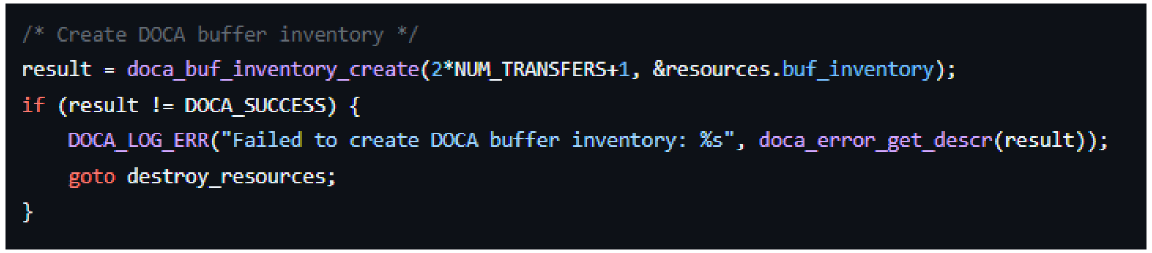
- I increased the size of the buffer inventory and I pre allocated 1000 buffers to not allocate one each time I want to submit a task. This uses a lot more memory and it is fixed that I cannot use memory after the 1000th buffer has ended, but it is faster. So, in the key value store application the buffers just have to be the size of the storage.
- To do that I needed to add local_bufs and remote_bufs arrays in the resources struct.

No changes to responder

Buffer pre allocation

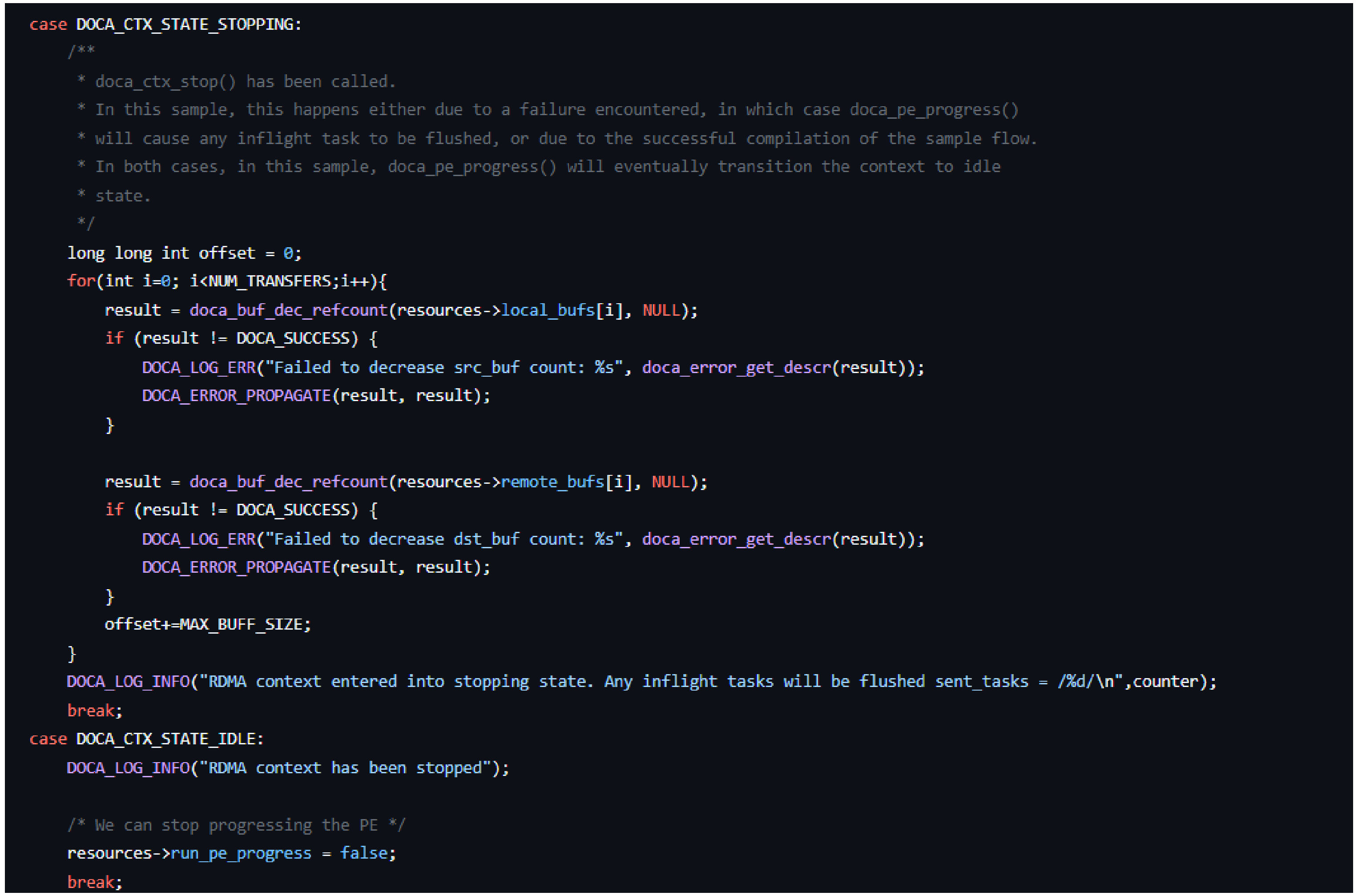


```
long long int offset = 0;
for(int i=0; i<NUM_TRANSFERS;i++){
    /* Add src buffer to DOCA buffer inventory from the remote mmap */
    tmp_result = doca_buf_inventory_buf_get_by_data(resources->buf_inventory,
                        resources->mmap,
                        resources->mmap_memrange + offset,
                        MAX_BUFF_SIZE,
                        &resources->local_bufs[i]);
    if (tmp_result != DOCA_SUCCESS) {
        DOCA_LOG_ERR("Failed to allocate DOCA buffer to DOCA buffer inventory: %s",
                doca_error_get_descr(tmp_result));
        // DOCA_ERROR_PROPAGATE(*first_encountered_error, tmp_result);
        return tmp_result;
    }

    tmp_result = doca_buf_inventory_buf_get_by_addr(resources->buf_inventory,
                        resources->remote_mmap,
                        resources->remote_mmap_range + offset,
                        MAX_BUFF_SIZE,
                        &resources->remote_bufs[i]);
    if (tmp_result != DOCA_SUCCESS) {
        DOCA_LOG_ERR("Failed to allocate DOCA buffer to DOCA buffer inventory: %s",
                doca_error_get_descr(tmp_result));
        // DOCA_ERROR_PROPAGATE(*first_encountered_error, tmp_result);
        return tmp_result;
    }
    if (tmp_result == DOCA_SUCCESS) {
        size_t buf_len;
        void *buf_data;
        doca_buf_get_len(resources->remote_bufs[i], &buf_len);
        doca_buf_get_data(resources->remote_bufs[i], &buf_data);
        DOCA_LOG_INFO("Remote buf %d created. Addr: %p, Len: %zu",
                    i, buf_data, buf_len);
    }
    offset = (offset + MAX_BUFF_SIZE)%PHYSICAL_BUFFER_SIZE;
```



```
/* Create DOCA buffer inventory */
result = doca_buf_inventory_create(2*NUM_TRANSFERS+1, &resources.buf_inventory);
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to create DOCA buffer inventory: %s", doca_error_get_descr(result));
    goto destroy_resources;
}
```

Freeing buffers in the end



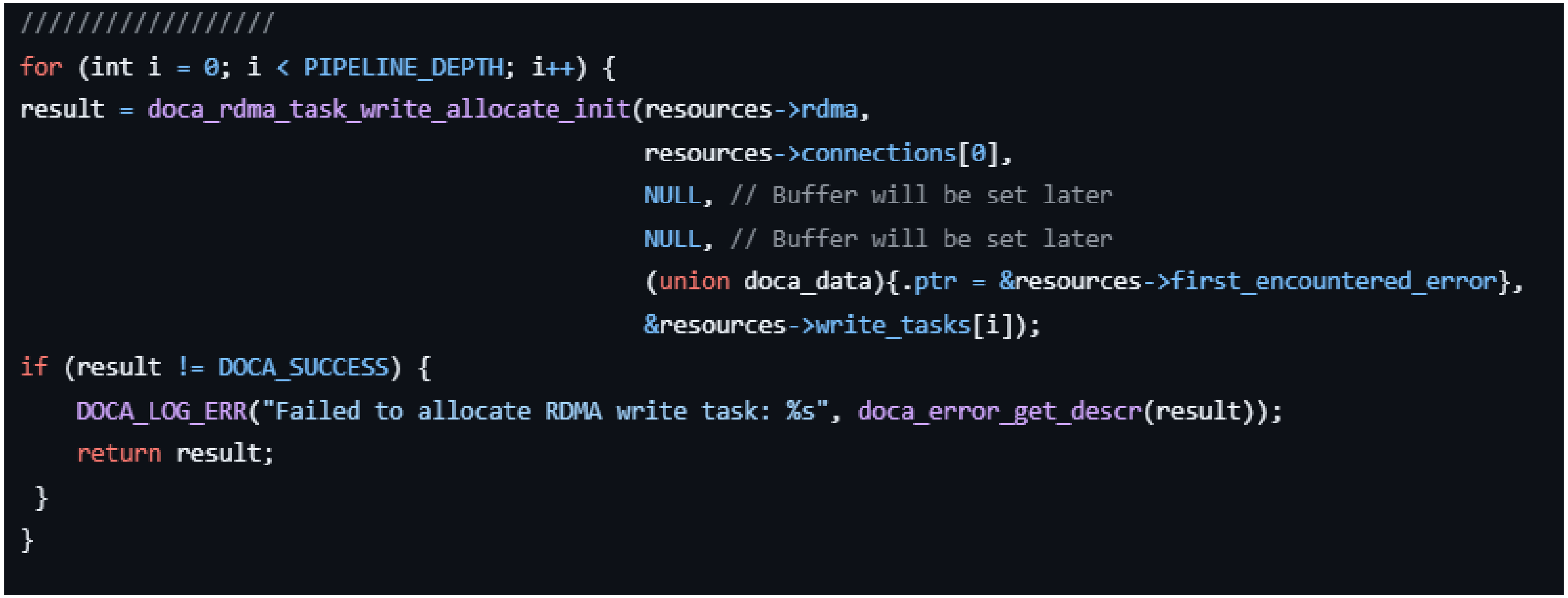
```
case DOCA_CTX_STATE_STOPPING:
    /**
     * doca_ctx_stop() has been called.
     * In this sample, this happens either due to a failure encountered, in which case doca_pe_progress()
     * will cause any inflight task to be flushed, or due to the successful compilation of the sample flow.
     * In both cases, in this sample, doca_pe_progress() will eventually transition the context to idle
     * state.
     */
    long long int offset = 0;
    for(int i=0; i<NUM_TRANSFERS;i++){
        result = doca_buf_dec_refcount(resources->local_bufs[i], NULL);
        if (result != DOCA_SUCCESS) {
            DOCA_LOG_ERR("Failed to decrease src_buf count: %s", doca_error_get_descr(result));
            DOCA_ERROR_PROPAGATE(result, result);
        }

        result = doca_buf_dec_refcount(resources->remote_bufs[i], NULL);
        if (result != DOCA_SUCCESS) {
            DOCA_LOG_ERR("Failed to decrease dst_buf count: %s", doca_error_get_descr(result));
            DOCA_ERROR_PROPAGATE(result, result);
        }
        offset+=MAX_BUFF_SIZE;
    }
    DOCA_LOG_INFO("RDMA context entered into stopping state. Any inflight tasks will be flushed sent_tasks = /%d/\n",counter);
    break;
case DOCA_CTX_STATE_IDLE:
    DOCA_LOG_INFO("RDMA context has been stopped");

    /* We can stop progressing the PE */
    resources->run_pe_progress = false;
    break;
```

Times

- Times:
  - Throughput: 47.6591 Gbps
  - Throughput: 60.8390 Gbps
  - Throughput: 43.9465 Gbps
  - Throughput: 35.2375 Gbps
  - Throughput: 64.5532 Gbps
  - Throughput: 48.2616 Gbps
- Average = 50.082817
- Standard deviation = 10.884085

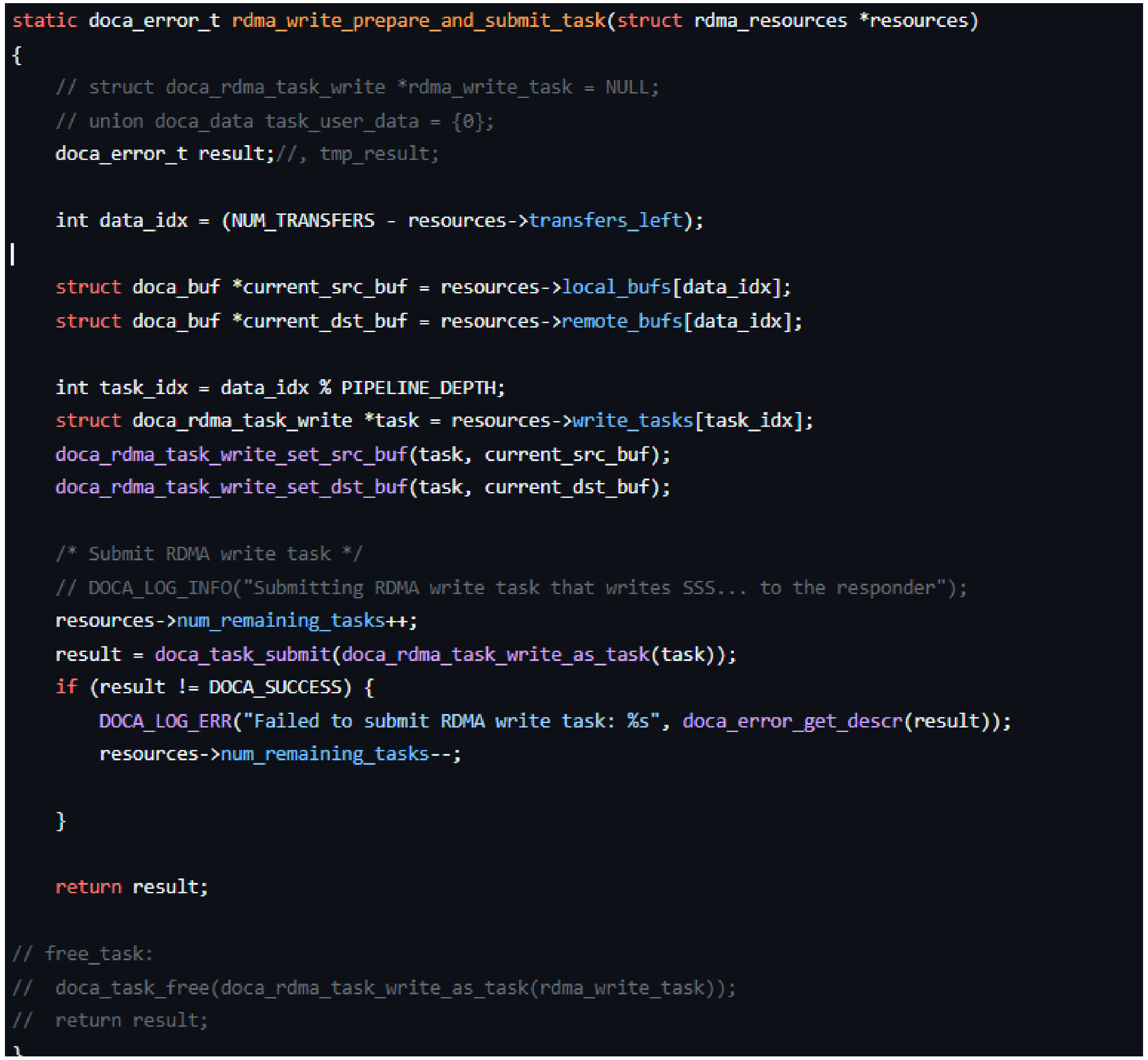
Version 8

Pre allocated task pool



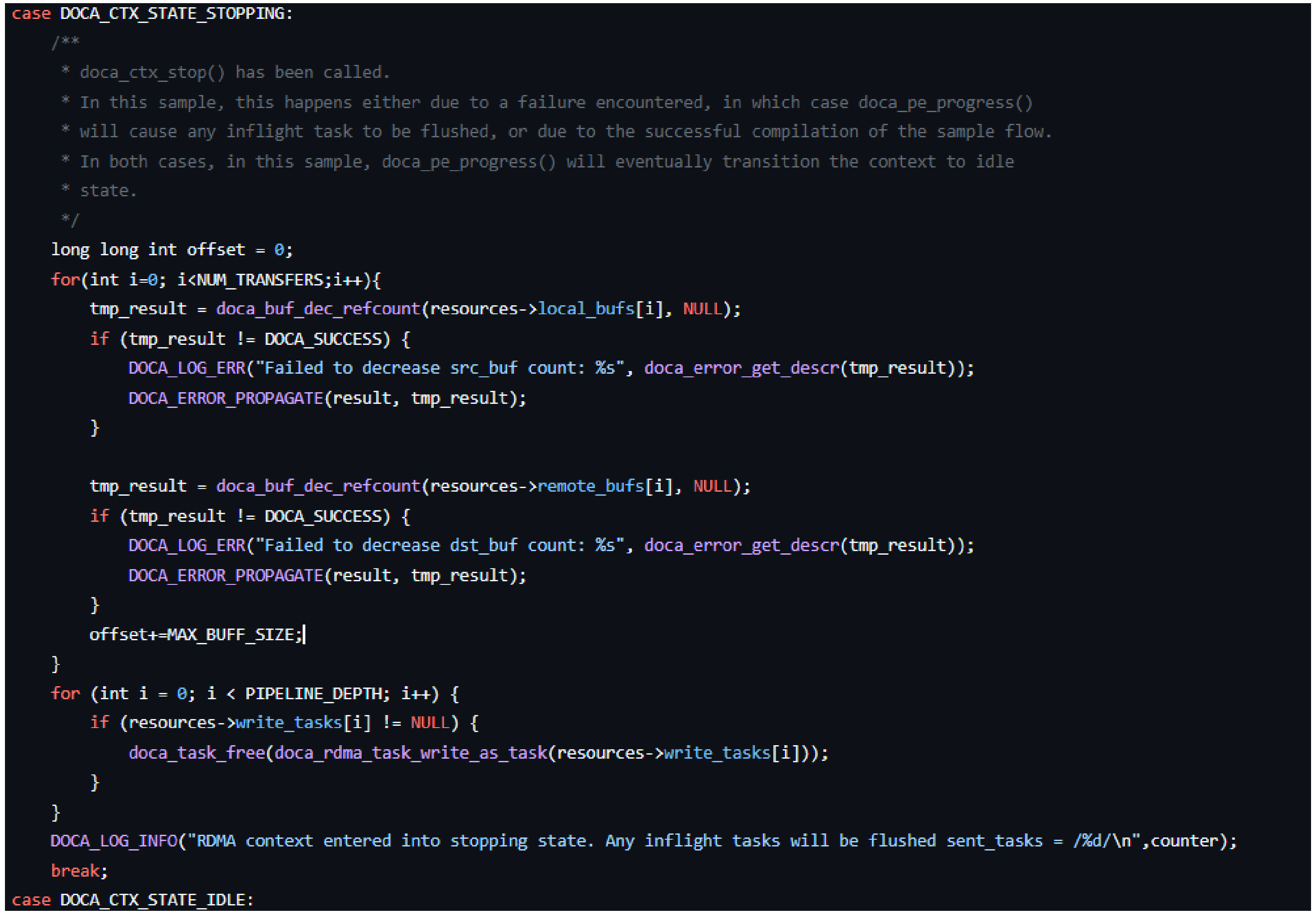
```
////////////////////
for (int i = 0; i < PIPELINE_DEPTH; i++) {
result = doca_rdma_task_write_allocate_init(resources->rdma,
                                            resources->connections[0],
                                            NULL, // Buffer will be set later
                                            NULL, // Buffer will be set later
                                            (union doca_data){.ptr = &resources->first_encountered_error},
                                            &resources->write_tasks[i]);
if (result != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to allocate RDMA write task: %s", doca_error_get_descr(result));
    return result;
 }
}
```

Reuse the pre allocated tasks



```c
static doca_error_t rdma_write_prepare_and_submit_task(struct rdma_resources *resources)
{
    // struct doca_rdma_task_write *rdma_write_task = NULL;
    // union doca_data task_user_data = {0};
    doca_error_t result;//, tmp_result;

    int data_idx = (NUM_TRANSFERS - resources->transfers_left);

    struct doca_buf *current_src_buf = resources->local_bufs[data_idx];
    struct doca_buf *current_dst_buf = resources->remote_bufs[data_idx];

    int task_idx = data_idx % PIPELINE_DEPTH;
    struct doca_rdma_task_write *task = resources->write_tasks[task_idx];
    doca_rdma_task_write_set_src_buf(task, current_src_buf);
    doca_rdma_task_write_set_dst_buf(task, current_dst_buf);

    /* Submit RDMA write task */
    // DOCA_LOG_INFO("Submitting RDMA write task that writes SSS... to the responder");
    resources->num_remaining_tasks++;
    result = doca_task_submit(doca_rdma_task_write_as_task(task));
    if (result != DOCA_SUCCESS) {
        DOCA_LOG_ERR("Failed to submit RDMA write task: %s", doca_error_get_descr(result));
        resources->num_remaining_tasks--;

    }

    return result;

// free_task:
//  doca_task_free(doca_rdma_task_write_as_task(rdma_write_task));
//  return result;
```

Task cleanup



```
case DOCA_CTX_STATE_STOPPING:
    /**
     * doca_ctx_stop() has been called.
     * In this sample, this happens either due to a failure encountered, in which case doca_pe_progress()
     * will cause any inflight task to be flushed, or due to the successful compilation of the sample flow.
     * In both cases, in this sample, doca_pe_progress() will eventually transition the context to idle
     * state.
     */
    long long int offset = 0;
    for(int i=0; i<NUM_TRANSFERS;i++){
        tmp_result = doca_buf_dec_refcount(resources->local_bufs[i], NULL);
        if (tmp_result != DOCA_SUCCESS) {
            DOCA_LOG_ERR("Failed to decrease src_buf count: %s", doca_error_get_descr(tmp_result));
            DOCA_ERROR_PROPAGATE(result, tmp_result);
        }

        tmp_result = doca_buf_dec_refcount(resources->remote_bufs[i], NULL);
        if (tmp_result != DOCA_SUCCESS) {
            DOCA_LOG_ERR("Failed to decrease dst_buf count: %s", doca_error_get_descr(tmp_result));
            DOCA_ERROR_PROPAGATE(result, tmp_result);
        }
        offset+=MAX_BUFF_SIZE;
    }
    for (int i = 0; i < PIPELINE_DEPTH; i++) {
        if (resources->write_tasks[i] != NULL) {
            doca_task_free(doca_rdma_task_write_as_task(resources->write_tasks[i]));
        }
    }
    DOCA_LOG_INFO("RDMA context entered into stopping state. Any inflight tasks will be flushed sent_tasks = /%d/\n",counter);
    break;
case DOCA_CTX_STATE_IDLE:
```

Times

- Times:
  - Throughput: 90.5575 Gbps
  - Throughput: 61.5395 Gbps
  - Throughput: 67.5940 Gbps
  - Throughput: 91.4312 Gbps
  - Throughput: 64.4800 Gbps
  - Throughput: 94.9797 Gbps
- Average = 78.430317
- Standard deviation = 15.409805

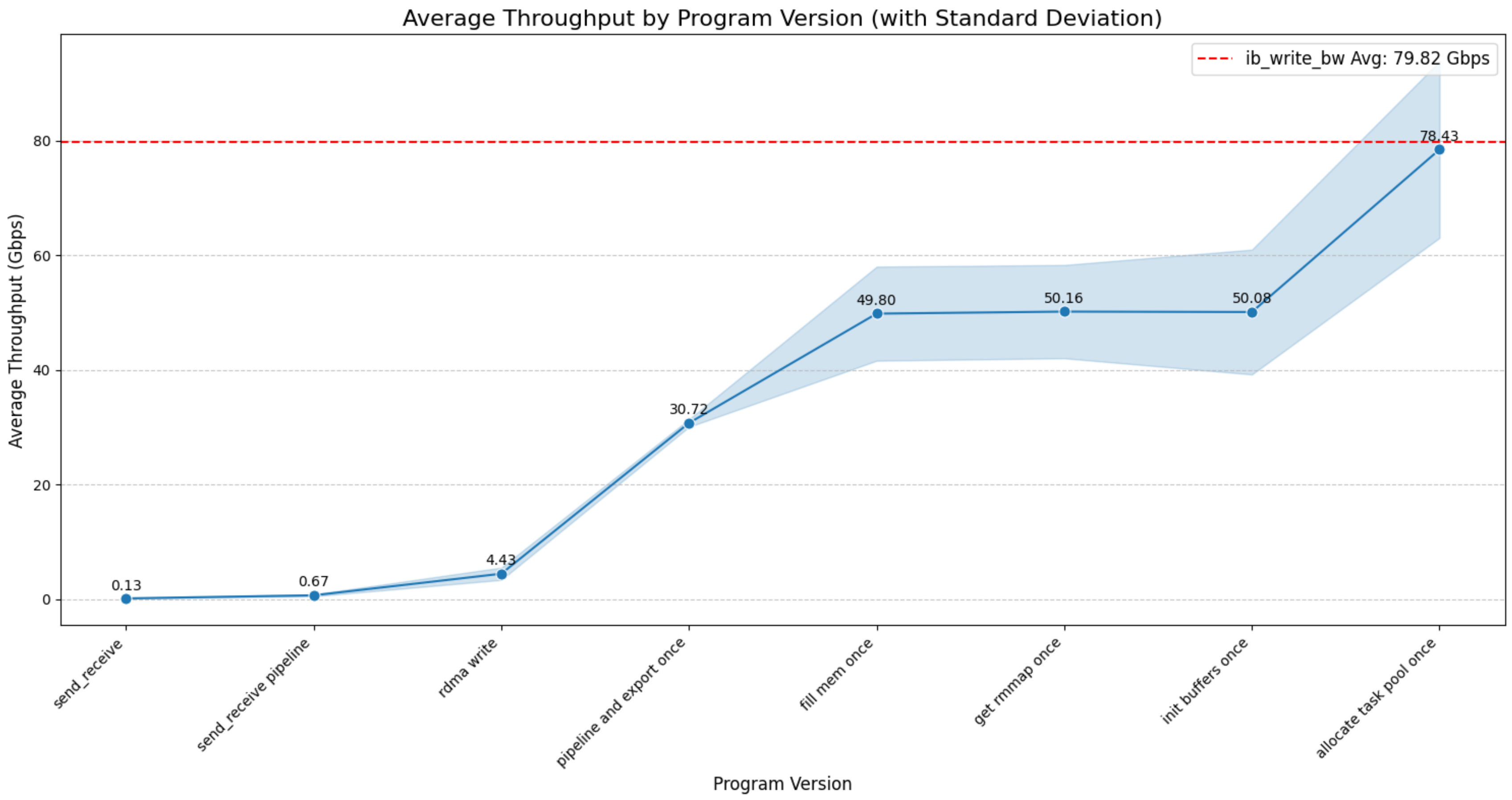
Ib_write_bw max throughput test

- To run the test run:
  - ib_write_bw, instruction to the server side
  - ib_write_bw <server_ip>, instruction to the client side

lb_write_bw max throughput test times

- The test indicated that the CPU was in a power-saving state so it could not reach its maximum potential.
- Times:
  - Throughput: 88.24 Gbps
  - Throughput: 89.87 Gbps
  - Throughput: 72.26 Gbps
  - Throughput: 69.93 Gbps
  - Throughput: 78.63 Gbps
  - Throughput: 80.0 Gbps
- Average = 79.821667
- Standard deviation = 8.103247

Times comparison



During the development of these versions some changes had to be made in both:

- rdma_common.c
- rdma_common.h

Changes in rdma_common.h

```
#define MEM_RANGE_LEN (1LL * 1024  * 1024 * 1024)
struct rdma_resources {
		// ... existing members

		struct timespec start_time;
		struct timespec end_time;
		int transfers_left;
		char *remote_mmap_range;
		size_t remote_mmap_range_len;
		struct doca_buf *local_bufs[1000];
		struct doca_buf *remote_bufs[1000];
		int cur_buf_idx;
		struct doca_rdma_task_write *write_tasks[128];
};
```

Changes in rdma_common.c

- Problem:
  - While developing the benchmarks, the DOCA driver gave a performance warning that memory range isn't aligned to 64B.
  - The problem was in the allocate_rdma_resources function, which was using calloc to get memory for the RDMA buffers.
  - The standard calloc (and malloc) functions do not guarantee 64-byte alignment, so they sometimes returned an unaligned address, triggering the warning.

Solution:

- I used posix_memalign instead of calloc.

```
void *memrange_ptr = NULL;
int ret = posix_memalign(&memrange_ptr, 64, MEM_RANGE_LEN);
```