# SwitchML

## Intel-KAUST-Microsoft

**Sep 19, 2021**

# CONTENTS


**1 SwitchML: Switch-Based Training Acceleration for Machine Learning** **1**
- 1.1 Getting started . . . 1
- 1.2 Repo organization . . . 1
- 1.3 Testing . . . 2
- 1.4 Publication . . . 2
- 1.5 Contributing . . . 2
- 1.6 The Team . . . 2
- 1.7 License . . . 3

**2 SwitchML Client Library** **5**
- 2.1 1. Backends . . . 5
- 2.2 2. Required Libraries . . . 5
- 2.3 2. Compiling the Library . . . 8
- 2.4 3. Using the library . . . 9

**3 SwitchML Client Library API** **11**
- 3.1 Class Hierarchy . . . 11
- 3.2 File Hierarchy . . . 11
- 3.3 Full API . . . 11

**4 SwitchML P4 program** **65**
- 4.1 1. Requirements . . . 65
- 4.2 2. Running the P4 program . . . 65
- 4.3 3. Design . . . 66
- 4.4 References . . . 66

**5 Switch Controller** **67**
- 5.1 Requirements . . . 67
- 5.2 Running the controller . . . 67

**6 Examples** **69**
- 6.1 1. Examples list . . . 69
- 6.2 2. Compiling . . . 69

**7 Benchmarks** **71**
- 7.1 1. Benchmarks list . . . 71
- 7.2 2. Compiling . . . 71

**8 Frameworks Integration** **73**

**9 Scripts** **75**

**10 Contributing** **77**
10.1 General Guidelines . . . 77
10.2 SwitchML Client Library . . . 78
10.3 SwitchML P4 Program . . . 80
10.4 SwitchML Controller . . . 80
10.5 Examples . . . 80
10.6 Benchmarks . . . 80
10.7 Documentation . . . 80

**Index** **81**

CHAPTER

ONE

# SWITCHML: SWITCH-BASED TRAINING ACCELERATION FOR MACHINE LEARNING

SwitchML accelerates the Allreduce communication primitive commonly used by distributed Machine Learning frameworks. It uses a programmable switch dataplane to perform in-network computation, reducing the volume of exchanged data by aggregating vectors (e.g., model updates) from multiple workers in the network. It provides an end-host library that can be integrated with ML frameworks to provide an efficient solution that speeds up training for a number of real-world benchmark models.

The switch hardware is programmed with a *P4 program* for the Tofino Native Architecture (TNA) and managed at runtime through a *Python controller* using BFRuntime. The *end-host library* provides simple APIs to perform Allreduce operations using different transport protocols. We currently support UDP through DPDK and RDMA UC. The library has already been integrated with ML frameworks as a NCCL plugin.

## 1.1 Getting started

To run SwitchML you need to:

- compile the P4 program and deploy it on the switch (see the *P4 code documentation*)
- run the Python controller (see the *controller documentation*)
- compile and run the end-host program using the end-host library (see the *library documentation*)

The *examples* folder provides simple programs that show how to use the APIs.

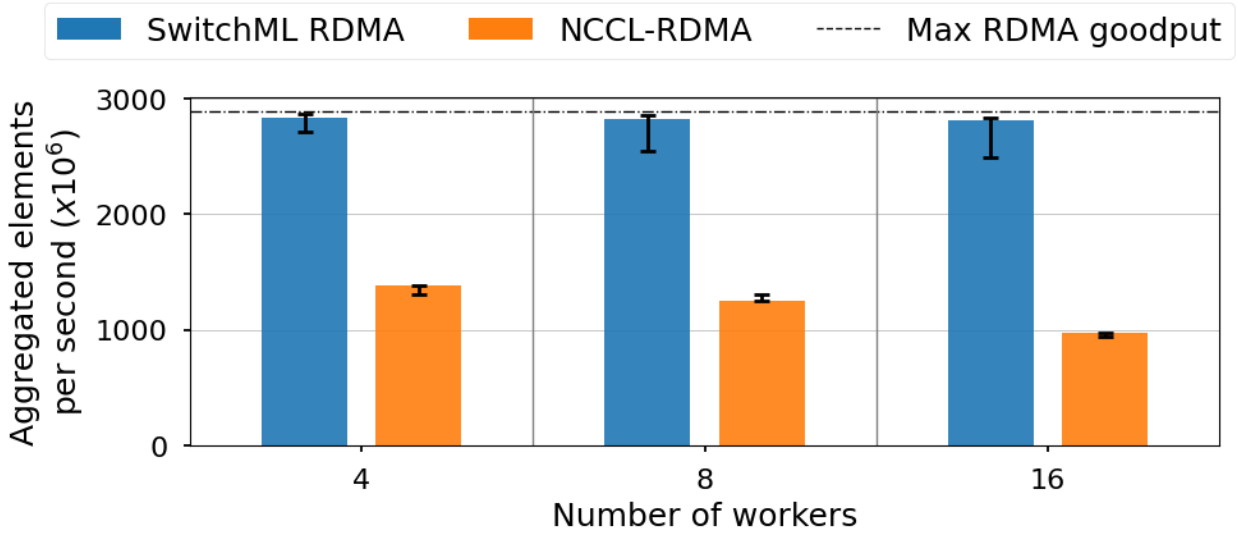
## 1.2 Repo organization

The SwitchML repository is organized as follows:

```
docs: project documentation
dev_root:
   p4: P4 code for TNA
   controller: switch controller program
   client_lib: end-host library
   examples: set of example programs
   benchmarks: programs used to test raw performance
   frameworks_integration: code to integrate with ML frameworks
   third_party: third party software
   protos: protobuf description for the interface between controller and end-host
   scripts: helper scripts
```

## 1.3 Testing

The *benchmarks* contain a benchmarks program that we used to measure SwitchML performances. In our experiments (see benchmark documentation for details) we observed a more than 2x speedup over NCCL when using RDMA. Moreover, differently from ring Allreduce, with SwitchML performance are constant with any number of workers.




## 1.4 Publication

> Scaling Distributed Machine Learning with In-Network Aggregation A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, P. Richtarik. In Proceedings of NSDI'21, Apr 2021.

## 1.5 Contributing

This project welcomes contributions and suggestions. To learn more about making a contribution to SwitchML, please see our *Contribution* page.

## 1.6 The Team

SwitchML is a project driven by the P4.org community and is currently maintained by Amedeo Sapio, Omar Alama, Marco Canini, Jacob Nelson.

## 1.7 License

SwitchML is released with an Apache License 2.0, as found in the LICENSE file.

CHAPTER

TWO

# SWITCHML CLIENT LIBRARY

The SwitchML client is a static library that bridges the gap between the end-hosts and the programmable switch through a simple to use API.

This document shows how to setup and use the library.

## 2.1 1. Backends

First of all you should know that the client library has multiple backends which perform the collective communication primitives.

### 2.1.1 1.1 Dummy Backend

The dummy backend is a backend that does not perform any actual communication but is just used for debugging purposes. It helps ensure that the software stack is operating correctly down to the backend.

### 2.1.2 1.2 DPDK Backend

The DPDK backend uses the DPDK library to perform collective operations with the UDP transport. Thus it supports all of the NICs and drivers that DPDK supports (we tested only Intel and Mellanox NICs so far).

### 2.1.3 1.3 RDMA Backend

The RDMA Backend uses ibverbs directly to perform communication using RDMA as a transport and it usually outperforms DPDK on more than 10Gbps NICs because of the additional hardware offloads. However, you must have a NIC that supports RDMA.

## 2.2 2. Required Libraries

Listed below are the system packages that are needed for the client library.

### 2.2.1 2.1 General requirements

These are dependencies that are required regardless of the backend you choose.

| Package (Debian/Ubuntu) | Tested Versions |
|---|---|
| gcc | 7.5.0-3ubuntu1~18.04 |
| make | 4.1-9.1ubuntu1 |
| build-essential | |
| libboost-program-options-dev | 1.65.1.0ubuntu1 |
| libgoogle-glog-dev | 0.3.5-1 |

On Debian/Ubuntu you can run the following command to install them:

```
sudo apt install -y \
gcc \
make \
libboost-program-options-dev \
libgoogle-glog-dev
```

### 2.2.2 1.2 DPDK Backend Requirements

These are dependencies that are required only for the DPDK backend.

| Package (Debian/Ubuntu) | Tested Versions |
|---|---|
| libnuma-dev | 2.0.11-2.1ubuntu0.1 |
| libibverbs-dev | 46mlnx1-1.46101 |
| libmnl-dev | 1.0.4-2 |
| autoconf | |
| libtool | |
| pkg-config | |
| cmake | 3.17.0 |
| libhugetlbfs-dev | |
| libssl-dev | |
| linux-headers | |
| linux-modules | |

The cmake version required to compile grpc must be at least 3.13 which is not available by default. Thus you will need to add kitware's repository to your build system by following this guide. Or you can choose to compile cmake from source.

On Debian/Ubuntu you can run the following command to install all dependencies (Assuming you added kitware's repo) :

```
sudo apt install -y \
libnuma-dev \
libibverbs-dev \
libhugetlbfs-dev \
libmnl-dev \
autoconf \
libtool \
pkg-config \
cmake \
libssl-dev \
linux-headers-$(uname -r) \
linux-modules-$(uname -r)
```

**Important** The DPDK backend requires root access. So whether you are running a benchmark, an example, or using it through pytorch, you must give your application root privileges.

### 2.2.3 1.3 RDMA Backend Requirements

These are dependencies that are required only for the RDMA backend.

| Package (Debian/Ubuntu) | Tested Versions |
|---|---|
| autoconf | |
| libtool | |
| pkg-config | |
| libibverbs-dev | 46mlnx1-1.46101 |
| cmake | 3.17.0 |
| libhugetlbfs-dev | |
| libssl-dev | |

The cmake version required to compile grpc must be at least 3.13 which is not available by default. Thus you will need to add kitware's repository to your build system by following this guide. Or you can choose to compile cmake from source.

On Debian/Ubuntu you can run the following command to install all dependencies (Assuming you added kitware's repo):

```
sudo apt install -y \
autoconf \
libtool \
pkg-config \
libibverbs-dev \
libhugetlbfs-dev \
cmake \
libssl-dev
```

**Important** The RDMA backend requires that you disable ICRC checking on the NIC that you will use. We provide a template for a script that does just that in the *scripts* directory.

## 2.3 2. Compiling the Library

To build the library with only the dummy backend for testing purposes you can simply run (Assuming you are in the client_lib directory)

```
make
```

To build the library with DPDK support, add `DPDK=1` to the make command.

```
make DPDK=1
```

To build the library with RDMA support, add `RDMA=1` to the make command.

```
make RDMA=1
```

By default the library will be found in:

```
dev_root/build/lib/libswitchml-client.a
```

Include files will be found in

```
dev_root/build/include
```

And finally the configuration file will be found in

```
dev_root/build/switchml.cfg
```

Read through the other options to control the build below.

### 2.3.1 2.1 Build Variables

The following variables can all be passed to the client_lib makefile to control the build.

| Variable | Type | Default | Usage |
|---|---|---|---|
| DEBUG | boolean | 0 | Disable optimizations, add debug symbols, and enable detailed debugging messages. |
| DPDK | boolean | 0 | Compile and include the dpdk backend. |
| RDMA | boolean | 0 | Compile and include the rdma backend. |
| DUMMY | boolean | 1 | Compile and include the dummy backend. |
| VCL | boolean | 1 | Compile with the vector class library (Used to speedup quantization on the CPU) |
| TIMEOUTS | boolean | 1 | Compile with timeouts and retransmissions support. |
| BUILDDIR | path | dev_root/build | Where to store generated objects/include files/libraries/binaries...etc. |
| GRPC_HOME | path | dev_root/third_party/grpc/build | Where to look for the GRPC installation |
| DPDK_HOME | path | dev_root/third_party/dpdk/build | Where to look for the DPDK installation |
| DPDK_SDK | path | dev_root/third_party/dpdk | Where to look for the DPDK SDK |
| VCL_HOME | path | dev_root/third_party/vcl | Where to look for the VCL headers |

## 2.4 3. Using the library

***Important*** *Before trying to use the library's API directly in your project, take a look at our* Frameworks Integration *directory to see if you can simply use one of the provided methods to integrate SwitchML into your DNN software stack.*

*What follows is intended to give you a high level overview of what needs to be done. For a more detailed step by step guide look at the* examples

After building the library and getting a copy of the include files, you can now use SwitchML in your project to perform collective communication. Follow these simple steps:

1. Edit your program
    1. Include the `context.h` file in your program.
    2. Call `switchml::Context::GetInstance()` to retrieve the singleton instance of the Context class.
    3. Call the `Start()` method of the context to start the SwitchML context.
    4. Use the API provided through the context instance reference.
    5. Call the `Stop()` method of the context to stop and cleanup the context.
2. Compile your program
    1. Add the following to your compiler arguments
        1. `-I path_to_includes`
        2. `-L path_to_library`
        3. `-l switchml_client`
3. Configure the SwitchML clients
    1. Before you can run your program you need to edit the configuration file that was generated after you built the library.
    2. After editing the `switchml.cfg` configuration file, copy it to where your program binary is.
4. Run your program

**Notes:**

- You can choose to create a Config object programmatically, edit its members, and pass it to the context as a parameter of the `Start()` method, instead of using the `switchml.cfg` file.
- For information on how to setup the switch, look at the *P4* and *controller* documentation.

CHAPTER

THREE

# SWITCHML CLIENT LIBRARY API

## 3.1 Class Hierarchy

## 3.2 File Hierarchy

## 3.3 Full API

### 3.3.1 Namespaces

#### Namespace switchml

**Contents**


- *Classes*
- *Enums*
- *Functions*
- *Typedefs*
- *Unions*


#### Classes


- *Struct BackendConfig*
- *Struct DpdkBackend::DpdkPacketHdr*
- *Struct DpdkBackend::E2eAddress*
- *Struct DpdkBackendConfig*
- *Struct DummyBackend::DummyPacket*
- *Struct GeneralConfig*
- *Struct JobSlice*
- *Struct RdmaBackendConfig*
- *Struct Tensor*

- *Struct TimeoutQueue::TQEntry*
- *Class Backend*
- *Class Barrier*
- *Class BypassPPP*
- *Class Config*
- *Class Context*
- *Class CpuExponentQuantizerPPP*
- *Class DpdkBackend*
- *Class DpdkMasterThread*
- *Class DpdkWorkerThread*
- *Class DummyBackend*
- *Class DummyWorkerThread*
- *Class FifoScheduler*
- *Class GrpcClient*
- *Class Job*
- *Class PrePostProcessor*
- *Class RdmaBackend*
- *Class RdmaConnection*
- *Class RdmaEndpoint*
- *Class RdmaWorkerThread*
- *Class Scheduler*
- *Class Stats*
- *Class TimeoutQueue*

### Enums

- *Enum AllReduceOperation*
- *Enum DataType*
- *Enum JobStatus*
- *Enum JobType*

### Functions

- *Function switchml::BindToCore*
- *Function switchml::ChangeMacEndianness*
- *Function switchml::DataTypeSize*
- *Function switchml::Execute*
- *Function switchml::GetCoresNuma*
- *Function switchml::GetDeviceNuma*
- *Function switchml::GIDToIPv4*
- *Function switchml::GIDToMAC*
- *Function switchml::IPv4ToGID*
- *Function switchml::Mac2Str(const uint64_t)*
- *Function switchml::Mac2Str(const rte_ether_addr)*
- *Function switchml::MACToGID*
- *Function switchml::Str2Mac*
- *Template Function switchml::ToHex*

### Typedefs

- *Typedef switchml::clock*
- *Typedef switchml::JobId*
- *Typedef switchml::Numel*
- *Typedef switchml::WorkerTid*

### Unions

- *Union ExtraJobInfo*

## 3.3.2 Classes and Structs

### Struct BackendConfig

- Defined in file_dev_root_client_lib_src_config.h

### Struct Documentation

**struct** switchml::**BackendConfig**
The struct that groups all backend related options.

#### Public Members

**struct** *DpdkBackendConfig* **dpdk**

**struct** *RdmaBackendConfig* **rdma**

### Struct DpdkBackend::DpdkPacketHdr

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_backend.h

### Nested Relationships

This struct is a nested type of *Class DpdkBackend*.

### Struct Documentation

**struct** switchml::*DpdkBackend*::**DpdkPacketHdr**
The switchml dpdk packet header.

#### Public Members

uint8_t **job_type_size**
This field is used to store both the job type and the packet's size enum or category. The 4 MSBs are for the job type and the 4 LSBs are for the size.

uint8_t **short_job_id**
The 8 LSBs of the id of the job associated with this packet. This is used by the client only to discard duplicates at the edge of switching from one job to another. Therefore we do not need the full length of the job id.

uint32_t **pkt_id**
An id to identify a packet within a job slice. This is used by the client only.

uint16_t **switch_pool_index**
The switch's pool/slot index.

A pool or a slot in the switch is what is used to store the values of a packet. Think of the switch as a large array of pools or slots. Each packet sent addresses a particular pool/slot.

The MSB of this field is used to alternate between two sets of pools/slots to have a shadow copy for switch retransmissions.

### Struct DpdkBackend::E2eAddress

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_backend.h

### Nested Relationships

This struct is a nested type of *Class DpdkBackend*.

### Struct Documentation

**struct** switchml::*DpdkBackend*::**E2eAddress**
A struct to store an end to end network address.

#### Public Members

uint64_t **mac**
An 8 bytes integer with the first 6 bytes representing the MAC address

uint32_t **ip**
A 4 byte integer representing the IP address.

uint16_t **port**
The 2 bytes integer representing the UDP port.

### Struct DpdkBackendConfig

- Defined in file_dev_root_client_lib_src_config.h

### Struct Documentation

**struct** switchml::**DpdkBackendConfig**
Configuration options specific to using the DPDK backend.

#### Public Members

uint16_t **worker_port**
The worker's udp port. No restrictions here just choose a port that is unused by any application

std::string **worker_ip_str**
Worker IP in the dotted decimal notation Choose the IP address for this worker and make sure its the one that corresponds to the correct network interface that you want to use for communication.

std::string **cores_str**
The DPDK core configuration As you know, each worker can have multiple threads/cores. Here we specify the specific cores that we want to use For Ex. cores = 10-13 will use 4 cores 10 through 13. cores = 10 will only use the core numbered 10). For best performance, you should use cores that are on the same NUMA node as the NIC. To do this do the following:

- Run `sudo lshw -class network -businfo` and take note of the PCIe identifier of the NIC (Ex. 0000:81:00.0).

- Run `lspci -s 0000:81:00.0 -vv | grep NUMA` to find out out the NUMA node at which this NIC resides.
- Run `lscpu | grep NUMA` to know the core numbers that also reside on the same NUMA node.

**make sure the number of cores you choose matches the number of worker threads in the general config**

std::string **extra_eal_options**
: These are extra options that will be passed to DPDK EAL. What we should include here is the PCIe identifier of the NIC using the -w option. (Ex. '-w 0000:81:00.0' ) to make sure that DPDK is using the right NIC. Otherwise you should find out the port id of the nic that you want.

uint16_t **port_id**
: Each NIC has an associated port id. Basically this is the index into the list of available NICs. If you've white listed the NIC that you want in extra_eal_options then you can always leave this as 0 since your chosen NIC will be the only one in the list.

uint32_t **pool_size**
: The size of the memory pool size for each worker thread. A memory pool is a chunk of memory from the huge pages which we use to allocate mbufs. Each worker thread will have its own memory pool for receiving mbufs (packets) and another for creating mbufs to be sent. Thus the total size of all memory pools can be calculated as pool_size*num_worker_threads*2. So just make sure you don't try to overallocate space that you don't have.

uint32_t **pool_cache_size**
: Each memory pool as described in pool_size has a dedicated software cache. How big do we want this to be? This value has strict restrictions from DPDK. If you don't know what you're doing you can leave it as it is.

uint32_t **burst_rx**
: What's the maximum number of packets that we retrieve from the NIC at a time.

uint32_t **burst_tx**
: What's the maximum number of packets that we push onto the NIC at a time.

uint32_t **bulk_drain_tx_us**
: Using what period in microseconds should we flush the transmit buffer?

### Struct DummyBackend::DummyPacket

- Defined in file_dev_root_client_lib_src_backends_dummy_dummy_backend.h

### Nested Relationships

This struct is a nested type of *Class DummyBackend*.

### Struct Documentation

**struct** switchml::*DummyBackend*::**DummyPacket**

A struct that describes the unit of transmission in the dummy backend (The *DummyPacket*).

A ***JobSlice*** is divided by the worker thread to multiple **DummyPacket* structs which then get sent then received using the backend.

#### Public Members

uint64_t **pkt_id**

A packet identifier unique only within a job slice. Accessed only by the worker thread that created the message. Can be calculated as packet offset from the job slice divided by the packet size

*JobId* **job_id**

The identifier of the job from which this message came from

*Numel* **numel**

The number of elements in the packet

*DataType* **data_type**

The data type of the elements

void ***entries_ptr**

Pointer to data that is supposed to be outstanding (in the network)

void ***extra_info_ptr**

Pointer to extra info that is supposed to be outstanding (in the network)

### Struct GeneralConfig

- Defined in file_dev_root_client_lib_src_config.h

### Struct Documentation

**struct** switchml::**GeneralConfig**

Struct that groups general configuration options that must always be configured.

#### Public Members

uint16_t **rank**

A unique identifier for a worker node. Like MPI ranks.

uint16_t **num_workers**

The number of worker nodes in the system

uint16_t **num_worker_threads**

The number of worker threads to launch for each node

uint32_t **max_outstanding_packets**

The maximum number of pending packets for this **worker** (Not worker thread).

This number is divided between worker threads. This means that each worker thread will first send its initial burst up to this number divided by num_worker_threads. Then sends new packets only after packets are received doing this until all packets have been sent.

If you have this set to 256 and num_worker_threads set to 8 then each worker thread will send up to 32 packets.

uint64_t **packet_numel**
The number of elements in a packet

std::string **backend**
Which backend should the SwitchML client use?. Choose from [‘dummy’, ‘dpdk’, ‘rdma’]. Make sure that the backend you choose has been compiled.

std::string **scheduler**
Which scheduler should we use to dispatch jobs to worker threads?. Choose from [‘fifo’].

std::string **prepostprocessor**
Which prepostprocessor should we use to load and unload the data into and from the network. Choose from [‘bypass’, ‘cpu_exponent_quantizer’]

bool **instant_job_completion**
If set to true then all jobs will be instantly completed regardless of the job type. This is used for debugging to disable all backend communication. The backend is still used to for setup and cleanup.

std::string **controller_ip_str**
The IP address of the machine that’s running the controller program. Note: This is not the same as the ip address that is passed to the switch_ip argument when starting the controller.

uint16_t **controller_port**
The port that the controller program is using. This is the value that you passed to the port argument when starting the controller.

double **timeout**
How much time in ms should we wait before we consider that a packet is lost.

Each worker thread creates a copy of this value at the start of working on a job slice. From that point the timeout value can be increased if the number of timeouts exceeds a threshold as a backoff mechanism.

uint64_t **timeout_threshold**
How many timeouts should occur before we double the timeout time?

uint64_t **timeout_threshold_increment**
By how much should we increment the threshold each time its exceeded. (Setting the bar higher to avoid doubling the timeout value too much)

### Struct JobSlice

- Defined in file_dev_root_client_lib_src_job.h

### Struct Documentation

**struct** switchml::**JobSlice**
A job slice that represents a part of a job.

This struct is what worker threads receive from the scheduler and what they work on.

#### Public Members

std::shared_ptr<*Job*> **job**
A reference to the original job of which this slice came from.

*Tensor* **slice**
The slice that the worker thread should work on

### Struct RdmaBackendConfig

- Defined in file_dev_root_client_lib_src_config.h

### Struct Documentation

**struct** switchml::**RdmaBackendConfig**
Configuration options specific to using the RDMA backend.

#### Public Members

uint32_t **msg_numel**
RDMA sends messages then the NIC splits a message into multiple packets. Thus the number of elements in a message must be a multiple of a packet's number of elements. This reduced the overheads involved in sending packet by packet. However, it also makes losses more costly for UC transport since the loss of a single packet will make us retransmit the whole message. Hence you should tweak this value until you find the sweet spot.

std::string **device_name**
The name of the Infiniband device to use. It will be something like `mlx5_0`. You can run the `ibv_devices` command to list your available devices.

uint16_t **device_port_id**
Each Infiniband device can have multiple ports. This value lets you choose a specific port. Use the `ibv_devinfo` command to list all ports in each device and see their id/index. Its the first number in the description of a port "port: 1" means you should use 1 for this variable.

uint16_t **gid_index**
Choose from the following: 0: RoCEv1 with MAC-based GID, 1:RoCEv2 with MAC-based GID, 2: RoCEv1 with IP-based GID, 3: RoCEv2 with IP-based GID

bool **use_gdr**
(Not implemented yet) Whether to try to use GPU Direct or not. In case the submitted job's data resides on the GPU, then using GPU Direct allows us to have our registerd buffer be also in GPU memory and directly send data from the GPU instead of having to copy it to a registered CPU buffer.

### Struct Tensor

- Defined in file_dev_root_client_lib_src_common.h

### Struct Documentation

**struct** switchml::**Tensor**

A struct to group up variables describing a tensor to be processed.

#### Public Functions

**inline** void **OffsetPtrs**(*Numel numel*)

A convenience function that offsets the tensor pointers by number of elements.

It casts the ptrs to the data_type then increments the pointers by numel argument. The member numel is untouched.

**Parameters** **numel** – **[in]** Number of **elements** to offset.

#### Public Members

void ***in_ptr**

Pointer to the input memory of the tensor. Any data changes are always written to the output. The input data is to be read from only !

void ***out_ptr**

Pointer to the output memory of the tensor

*Numel* **numel**

Number of **elements** in the tensor. (Not the size)

*DataType* **data_type**

The numerical data type of the elements in the tensor

### Struct TimeoutQueue::TQEntry

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_timeout_queue.h

### Nested Relationships

This struct is a nested type of *Class TimeoutQueue*.

### Struct Documentation

**struct** switchml::*TimeoutQueue*::**TQEntry**

A struct representing a single entry in the *TimeoutQueue*.

#### Public Functions

**TQEntry**()

**~TQEntry**() = default

**TQEntry**(*TQEntry* **const**&) = default

void **operator=**(*TQEntry* **const**&) = delete

**TQEntry**(*TQEntry*&&) = default

*TQEntry* &**operator=**(*TQEntry*&&) = default

#### Public Members

bool **valid**

Whether this entry is just a place holder or an actual entry pushed by the user.

int **next**

The index of the next entry

int **previous**

The index of the previous entry

*TimePoint* **timestamp**

The time at which this entry was pushed

### Class Backend

- Defined in file_dev_root_client_lib_src_backend.h

### Inheritance Relationships

### Derived Types

- public switchml::DpdkBackend (*Class DpdkBackend*)
- public switchml::DummyBackend (*Class DummyBackend*)
- public switchml::RdmaBackend (*Class RdmaBackend*)

## Class Documentation

**class** switchml::**Backend**

An interface that describes the backend.

A backend is the class responsible for creating worker threads and actually carrying out the jobs submitted by performing the communication.

Subclassed by *switchml::DpdkBackend*, *switchml::DummyBackend*, *switchml::RdmaBackend*

### Public Functions

**~Backend**() = default

**Backend**(*Backend* **const**&) = delete

void **operator=**(*Backend* **const**&) = delete

**Backend**(*Backend*&&) = default

*Backend* &**operator=**(*Backend*&&) = default

**virtual** void **SetupWorker**() = 0

Initializes backend specific variables and starts worker threads.

**See** *CleanupWorker()*

**virtual** void **CleanupWorker**() = 0

Cleans up all worker state and **waits** for the worker threads to exit.

**See** *SetupWorker()*

### Public Static Functions

**static** std::unique_ptr<*Backend*> **CreateInstance**(*Context* &*context*, *Config* &*config*)

Factory function to create a backend instance based on the configuration passed.

**Parameters**

- **context** – **[in]** a reference to the switchml context.
- **config** – **[in]** a reference to the switchml configuration.

**Returns** std::unique_ptr<Backend> a unique pointer to the created backend object.

### Protected Functions

**Backend**(*Context* &*context*, *Config* &*config*)

Initializes the members with the passed references.

Must be called explicitly by all subclass constructors.

**Parameters**

- **context** – **[in]** The context
- **config** – **[in]** The context configuration.

#### Protected Attributes

*Context* &**context_**
A reference to the context

*Config* &**config_**
A reference to the context configuration

### Class Barrier

- Defined in file_dev_root_client_lib_src_utils.h

### Class Documentation

**class** switchml::**Barrier**
A class that implements a simple thread barrier. Simply create an instance that is visible to using threads then from each thread call the wait function.

#### Public Functions

**Barrier**(**const** int *num_participants*)
Construct a new *Barrier* object.

**Parameters** **num_participants** – **[in]** Number of threads that will use the barrier

**~Barrier**()
Call *Destroy()* just in case it hasn't been called and some threads are waiting.

**See** *Destroy()*

**Barrier**(*Barrier* **const**&) = delete

void **operator=**(*Barrier* **const**&) = delete

**Barrier**(*Barrier*&&) = default

*Barrier* &**operator=**(*Barrier*&&) = default

void **Wait**()
Block the thread until all other participating threads arrive at the barrier.

void **Destroy**()
Wakeup all waiting threads and make this barrier unusable.

### Class BypassPPP

- Defined in file_dev_root_client_lib_src_prepostprocessors_bypass_ppp.h

### Inheritance Relationships

### Base Type

- `public switchml::PrePostProcessor` (*Class PrePostProcessor*)

### Class Documentation

**class** `switchml::BypassPPP` : **public** switchml::*PrePostProcessor*

A class that ignores prepostprocessing completely and just serves as a placeholder.

It is used for debugging and measuring performance without any prepostprocessing. It consists of mostly empty inline functions that will most likely be simply compiled away.

#### Public Functions

**inline BypassPPP**(*Config* &*config*, *WorkerTid* *worker_tid*, *Numel* *ltu_size*, *Numel* *batch_num_ltus*)

Calls the super class constructor.

**Parameters**

- **config** – **[in]** A reference to the context's configuration.
- **worker_tid** – **[in]** The worker thread that this prepostprocessor belongs to.
- **ltu_size** – **[in]** The size in bytes of the logical transmission unit used by the backend.
- **batch_num_ltus** – **[in]** How many LTUs constitute a batch.

**~BypassPPP**() = default

**BypassPPP**(*BypassPPP* **const**&) = delete

void **operator=**(*BypassPPP* **const**&) = delete

**BypassPPP**(*BypassPPP*&&) = default

*BypassPPP* &**operator=**(*BypassPPP*&&) = default

**inline virtual** uint64_t **SetupJobSlice**(*JobSlice* **job_slice*) **override**

Compute the number of LTUs needed.

**Parameters** **job_slice** – **[in]** A pointer to the job slice currently being worked on by the worker thread.

**Returns** uint64_t the number of transmission units that prepostprocessor will need to be sent and received by the backend.

**inline virtual** bool **NeedsExtraBatch**() **override**

always return false

**Returns** true Never

**Returns** false Always

**inline** void **PreprocessSingle**(__attribute__((unused)) uint64_t *pkt_id*, __attribute__((unused)) void **entries_ptr*, __attribute__((unused)) void **extra_info*) **override**

Do nothing.

**Parameters**

- **pkt_id** – ignored
- **entries_ptr** – ignored
- **extra_info** – ignored

**inline** void **PostprocessSingle**(__attribute__((unused)) uint64_t *pkt_id*, __attribute__((unused)) void **entries_ptr*, __attribute__((unused)) void **extra_info*) **override**

Do nothing.

**Parameters**

- **pkt_id** – ignored
- **entries_ptr** – ignored
- **extra_info** – ignored

**inline virtual** void **CleanupJobSlice**() **override**

Do nothing.

## Class Config

- Defined in file_dev_root_client_lib_src_config.h

## Class Documentation

**class** switchml::**Config**

A class that is responsible for parsing and representing all configurable options for SwitchML.

### Public Functions

**Config**() = default

**~Config**() = default

**Config**(*Config* **const**&) = default

void **operator=**(*Config* **const**&)

**Config**(*Config*&&) = default

*Config* &**operator=**(*Config*&&) = default

bool **LoadFromFile**(std::string *path* = "")

Read and parse the configuration file.

**Parameters** **path** – **[in]** the path of the configuration file or nullptr. If the path was ommited then the function looks for the file in the following default paths in order: 1- /etc/switchml.cfg 2- ./switchml.cfg 3- ./switchml-<hostname>.cfg (Ex. ./switchml-node12.config)

**Returns** loading was successfull

**Returns** loading failed.

void **Validate**()

Make sure configuration values are valid.

If a misconfiguration is fatal then it shuts the program down.

void **PrintConfig**()
Print all configuration options.

### Public Members

**struct** *GeneralConfig* **general_**
General configuration options

**struct** *BackendConfig* **backend_**
*Backend* specific configuration options

## Class Context

- Defined in file_dev_root_client_lib_src_context.h

## Class Documentation

**class** switchml::**Context**
Singleton class that represents the SwitchML API.

This is the starting point for all SwitchML operations. Simply create a context, start the context, do your operations, stop the context.

### Public Types

**enum ContextState**
An enum to describe the context's state.

The context goes through all states sequentially during its lifetime.

*Values:*

**enumerator CREATED**
Was just constructed. must call *Start()*.

**enumerator STARTING**
In the process of initializing and starting.

**enumerator RUNNING**
Running and ready to receive job requests.

**enumerator STOPPING**
In the process of shutting down.

**enumerator STOPPED**
Shutdown completed.

### Public Functions

**Context**(*Context* **const**&) = delete

void **operator=**(*Context* **const**&) = delete

**Context**(*Context*&&) = delete

*Context* &**operator=**(*Context*&&) = delete

bool **Start**(*Config* **config* = NULL)

Perform all needed initializations to make SwitchML ready to be used through the context api.

The function performs all of the following:

- Parse configuration files
- Initialize and allocate variables and structures.
- Setup the backend (This includes starting worker threads)

**See** *Stop()*

**Parameters** **config** – **[in]** A pointer to a configuration object to use. If the argument is not passed then the configuration will be created and loaded from the default configuration paths using *Config::LoadFromFile()*.

**Returns** true Initialization was successfull and you can start using the context.

**Returns** false Initialization failed. Any subsequent calls to the context api will have undefined behavior.

void **Stop**()

Performs all needed steps to stop switchml and cleanup all of its state.

The function performs all of the following:

- Clean up the backend (This includes stopping worker threads and **waiting** for them)
- Clean up all dynamically allocated memory.

**See** *Start()*

std::shared_ptr<*Job*> **AllReduceAsync**(void **in_ptr*, void **out_ptr*, uint64_t *numel*, *DataType* *data_type*, *AllReduceOperation* *all_reduce_operation*)

The function will submit an all reduce *Job* to the *Context Scheduler* then return immedietly.

The reduced tensor will be stored inplace in the same buffer provided. Consider calling WaitForCompletion or GetJobStatus on the returned *Job* object reference to make sure that it completed.

**See** *AllReduce()*

**Parameters**

- **in_ptr** – **[in]** Pointer to the memory where to read data
- **out_ptr** – **[in]** Pointer to the memory where to write processed data (The results)
- **numel** – **[in]** Number of elements (Not size)
- **data_type** – **[in]** The type of the data (FLOAT32, INT32).

- **all_reduce_operation** – **[in]** what kind of all reduce operation do you want to perform?

**Returns** std::shared_ptr<Job> A shared pointer to the job that was submitted.

std::shared_ptr<*Job*> **AllReduce**(void **in_ptr*, void **out_ptr*, uint64_t *numel*, *DataType* *data_type*, *AllReduceOperation* *all_reduce_operation*)

Convenience function equivelant to calling AllReduceAsync then waiting on the returned job reference.

**See** *AllReduceAsync()*

**See** *Job::WaitToComplete()*

void **WaitForAllJobs**()

Blocks the calling thread until SwitchML finishes all submited work.

Finishing includes failing and dropping the job. So the job status should be checked.

**See** *Job::WaitToComplete()*

*ContextState* **GetContextState**()

Get the current *Context* State.

**Returns** ContextState

**const** *Config* &**GetConfig**()

Get a constant reference to the active configuration.

**Returns** const *Config*&

*Stats* &**GetStats**()

Get a reference to the statistics object used.

**Returns** *Stats*&

### Public Static Functions

**static** *Context* &**GetInstance**()

Gets a reference to the single *Context* object.

A new instance is created (Constructor is called) when you call this function for the first time. Subsequent calls will retrieve the same context object. The instance only gets destroyed (Destructor is called) when the program exits like the default with any static object.

**Returns** *Context*& A reference to the context object.

## Class CpuExponentQuantizerPPP

- Defined in file_dev_root_client_lib_src_prepostprocessors_cpu_exponent_quantizer_ppp.h

### Inheritance Relationships

### Base Type

- `public switchml::PrePostProcessor` (*Class PrePostProcessor*)

### Class Documentation

**class** `switchml::`**CpuExponentQuantizerPPP** : **public** switchml::*PrePostProcessor*

A class that implements the switchml exponent quantization scheme using CPU instructions.

#### Public Functions

**CpuExponentQuantizerPPP**(*Config* *&config*, *WorkerTid* *worker_tid*, *Numel* *ltu_size*, *Numel* *batch_num_ltus*)

Calls the super class constructor and initialize this class's members.

**Parameters**

- **config** – **[in]** A reference to the context's configuration.
- **worker_thread_id** – **[in]** The worker thread that this prepostprocessor belongs to.
- **ltu_size** – **[in]** The size in bytes of the logical transmission unit used by the backend.
- **batch_num_ltus** – **[in]** How many LTUs constitute a batch.

**~CpuExponentQuantizerPPP**()

Calls *CleanupJobSlice()* to make sure that any dynamically allocated memory is released.

**See** *CleanupJobSlice()*

**CpuExponentQuantizerPPP**(*CpuExponentQuantizerPPP* **const**&) = delete

void **operator=**(*CpuExponentQuantizerPPP* **const**&) = delete

**CpuExponentQuantizerPPP**(*CpuExponentQuantizerPPP*&&) = default

*CpuExponentQuantizerPPP* &**operator=**(*CpuExponentQuantizerPPP*&&) = default

**virtual** uint64_t **SetupJobSlice**(*JobSlice* **job_slice*) **override**

Prepare the prepostprocessor's internal variables for this job slice.

This must be called as soon as the worker thread receives a job slice.

**See** *CleanupJobSlice()*

**Parameters** **job_slice** – **[in]** A pointer to the job slice currently being worked on by the worker thread.

**Returns** uint64_t the number of transmission units that prepostprocessor will need to be sent and received by the backend.

**virtual** bool **NeedsExtraBatch**() **override**

Check whether the currently running job slice needs an extra batch or not.

**Returns** true if the data type is float32

**Returns** false otherwise

**virtual** void **PreprocessSingle**(uint64_t *ltu_id*, void **entries_ptr*, void **exponent_ptr*) **override**

Preprocess a tensor converting it to switchml's representation and loading it into the backend's buffers.

**See** *PostprocessSingle()*

**Parameters**

- **ltu_id** – **[in]** The id of the logical transmission unit to be preprocessed within the current job slice. ltu_id will be used to compute the offset into the job slice ltu_id * ltu_size.
- **entries_ptr** – **[out]** A pointer to where we will store the quantized payload.
- **exponent_ptr** – **[out]** A pointer to where we will store the exponent in the packet.

**virtual** void **PostprocessSingle**(uint64_t *ltu_id*, void **entries_ptr*, void **exponent_ptr*) **override**

Postprocess a tensor converting it to the client's representation and loading it into the client's buffers.

**See** *PreprocessSingle()*

**Parameters**

- **ltu_id** – **[in]** The id of the logical transmission unit to be preprocessed within the current job slice. ltu_id will be used to compute the offset into the job slice ltu_id * ltu_size.
- **entries_ptr** – **[in]** A pointer to where we will read the received payload from.
- **exponent_ptr** – **[in]** A pointer to where we will read the exponent from.

**virtual** void **CleanupJobSlice**() **override**

Cleans up all internal structures and release any dynamically allocated memory associated with the job slice.

**See** *SetupJobSlice()*

### Class DpdkBackend

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_backend.h

### Nested Relationships

### Nested Types

- *Struct DpdkBackend::DpdkPacketHdr*
- *Struct DpdkBackend::E2eAddress*

### Inheritance Relationships

### Base Type

- `public switchml::Backend` (*Class Backend*)

### Class Documentation

**class** `switchml::DpdkBackend` : **public** switchml::*Backend*

The backend that represents the dpdk version of switchml.

#### Public Types

**typedef** int32_t **DpdkPacketElement**

A type representing a single element in the packet

#### Public Functions

**struct switchml::DpdkBackend::DpdkPacketHdr __attribute__ ((__packed__))**

**DpdkBackend**(*Context* &*context*, *Config* &*config*)

Call the super class constructor.

**Parameters**

- **context** – **[in]** The context
- **config** – **[in]** The context configuration.

**~DpdkBackend**()

**DpdkBackend**(*DpdkBackend* **const**&) = delete

void **operator=**(*DpdkBackend* **const**&) = delete

**DpdkBackend**(*DpdkBackend*&&) = default

*DpdkBackend* &**operator=**(*DpdkBackend*&&) = default

**virtual** void **SetupWorker**() **override**

Creates and starts the dpdk master thread.

Which in turn initializes the DPDK EAL and creates and starts all worker threads.

**See** *DpdkMasterThread::operator()()*

**See** *DpdkWorkerThread::operator()()*

**See** *CleanupWorker()*

**virtual** void **CleanupWorker**() **override**

Waits for the dpdk master thread to exit.

**See** *SetupWorker()*

void **SetupSwitch**()
Contacts the controller using the GRPC Client and tells it to create a UDP session.

This must not be called until the IP and MAC addresses of the switch and worker have been filled correctly.

**struct** *E2eAddress* &**GetSwitchE2eAddr**()
Get a reference to the switch end to end address in big endian.

The reference can be used to modify the address.

**Returns** struct *E2eAddress*& a reference to the address object.

**struct** *E2eAddress* &**GetWorkerE2eAddr**()
Get a reference to the worker end to end address in big endian.

The reference can be used to modify the address.

**Returns** struct *E2eAddress*& a reference to the address object.

std::vector<*DpdkWorkerThread*> &**GetWorkerThreads**()
Get a list of the worker threads.

**Returns** std::vector<DpdkWorkerThread>

### Public Members

**struct switchml::DpdkBackend::E2eAddress __attribute__**

**struct DpdkPacketHdr**
The switchml dpdk packet header.

#### Public Members

uint8_t **job_type_size**
This field is used to store both the job type and the packet's size enum or category. The 4 MSBs are for the job type and the 4 LSBs are for the size.

uint8_t **short_job_id**
The 8 LSBs of the id of the job associated with this packet. This is used by the client only to discard duplicates at the edge of switching from one job to another. Therefore we do not need the full length of the job id.

uint32_t **pkt_id**
An id to identify a packet within a job slice. This is used by the client only.

uint16_t **switch_pool_index**
The switch's pool/slot index.

A pool or a slot in the switch is what is used to store the values of a packet. Think of the switch as a large array of pools or slots. Each packet sent addresses a particular pool/slot.

The MSB of this field is used to alternate between two sets of pools/slots to have a shadow copy for switch retransmissions.

**struct E2eAddress**
A struct to store an end to end network address.

#### Public Members

uint64_t **mac**
An 8 bytes integer with the first 6 bytes representing the MAC address

uint32_t **ip**
A 4 byte integer representing the IP address.

uint16_t **port**
The 2 bytes integer representing the UDP port.

### Class DpdkMasterThread

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_master_thread.h

### Class Documentation

**class** `switchml::`**DpdkMasterThread**
A class that represents a single dpdk master thread.

A single instance is created of this thread. The thread is responsible for creating, starting, and managing all of the dpdk worker threads.

**See** *DpdkWorkerThread*

#### Public Functions

**DpdkMasterThread**(*Context* &*context*, *DpdkBackend* &*backend*, *Config* &*config*)
Initialize all members.

**Parameters**

- **context** – **[in]** a reference to the switchml context.
- **backend** – **[in]** a reference to the created dpdk backend.
- **config** – **[in]** a reference to the context configuration.

**~DpdkMasterThread**()
Deletes the reference to the system thread.

**DpdkMasterThread**(*DpdkMasterThread* **const**&) = default

void **operator=**(*DpdkMasterThread* **const**&) = delete

**DpdkMasterThread**(*DpdkMasterThread*&&) = default

*DpdkMasterThread* &**operator=**(*DpdkMasterThread*&&) = default

void **operator()**()
This is the point of entry function for the thread.

The function starts by initializing EAL then starting worker threads. Then the master thread itself becomes a worker thread. Finally, when the master thread finishes its worker thread function it waits for other threads and cleans up before exiting.

void **Start**()
Start the thread.

void **Join**()
Wait for the thread to exit and delete its system reference.

### Class DpdkWorkerThread

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_worker_thread.h

### Class Documentation

**class** switchml::**DpdkWorkerThread**
A class that represents a single dpdk worker thread.

A worker thread constantly asks the context for work and carries it out.

Multiple instances of this class is typically created depending on the number of cores in the configuration. This class has no Start and Join functions as other typical thread classes in the client library. This is because starting and joining the DPDK worker thread is handled by DPDK itself.

#### Public Functions

**DpdkWorkerThread**(*Context* &*context*, *DpdkBackend* &*backend*, *Config* &*config*)
Initialize all members and instantiate the prepostprocessor to be used by the worker thread.

**Parameters**

- **context** – **[in]** a reference to the switchml context.
- **backend** – **[in]** a reference to the created dpdk backend.
- **config** – **[in]** a reference to the context configuration.

**~DpdkWorkerThread**()

**DpdkWorkerThread**(*DpdkWorkerThread* **const**&) = delete

void **operator=**(*DpdkWorkerThread* **const**&) = delete

**DpdkWorkerThread**(*DpdkWorkerThread*&&) = default

*DpdkWorkerThread* &**operator=**(*DpdkWorkerThread*&&) = default

void **operator()**()
This is the point of entry function for the thread.

#### Public Members

**const** *WorkerTid* **tid_**
Worker thread id

### Class DummyBackend

- Defined in file_dev_root_client_lib_src_backends_dummy_dummy_backend.h

### Nested Relationships

### Nested Types

- *Struct DummyBackend::DummyPacket*

### Inheritance Relationships

### Base Type

- `public switchml::Backend` (*Class Backend*)

### Class Documentation

**class** `switchml::`**DummyBackend** : **public** switchml::*Backend*

A backend for debugging which simulates communication by sleeping.

It allows us to test the correctness of all components without having to deal with the complexities of a real backend and without performing any actual communication. The backend launches worker threads and sleeps when a send or receive is called. The sleeping duration is determined by the dummy bandwidth and the size of the tensor. The bandwidth is configurable through the configuration file.

#### Public Functions

**DummyBackend**(*Context* &*context*, *Config* &*config*)

Initialize members and allocate worker_threads and pending_messages arrays.

**Parameters**

- **context** – **[in]** a reference to the switchml context.
- **config** – **[in]** a reference to the switchml configuration.

**~DummyBackend**()

Free worker_threads and pending_messages arrays.

**DummyBackend**(*DummyBackend* **const**&) = delete

void **operator=**(*DummyBackend* **const**&) = delete

**DummyBackend**(*DummyBackend*&&) = default

*DummyBackend* &**operator=**(*DummyBackend*&&) = default

**virtual** void **SetupWorker**() **override**

Creates and starts worker threads.

**See** *CleanupWorker()*

**virtual** void **CleanupWorker**() **override**
Stops worker threads.

**See** *SetupWorker()*

void **SetupWorkerThread**(*WorkerTid worker_thread_id*)
Does nothing.

void **CleanupWorkerThread**(*WorkerTid worker_thread_id*)
Does nothing.

void **SendBurst**(*WorkerTid worker_thread_id*, **const** std::vector<*DummyPacket*> &*packets_to_send*)
Sends a burst of packets specific to a worker thread.

This is a generic function that could be used to send a single message or multiple packets at once. The function sleeps for a period equal to all packets sizes divided by the dummy backend bandwidth to simulate network sending.

The sent packets are stored internally so that they can later be retrieved by *ReceiveBurst()*

**See** *ReceiveBurst()*

**Parameters**

- **worker_thread_id** – **[in]** The id of the calling worker thread.
- **packets_to_send** – **[in]** A vector of dummy packets to send.

void **ReceiveBurst**(*WorkerTid worker_thread_id*, std::vector<*DummyPacket*> &*packets_received*)
Receives a burst of packets specific to a worker thread.

This function returns a random number of packets from the packets that the worker has sent using *SendBurst()* The packets can be received out of order to simulate a real network. Before the packets are returned, the elements are multiplied by the number of workers to simulate that an AllReduce Sum opearation took place.

**See** *SendBurst()*

**Parameters**

- **worker_thread_id** – **[in]** The id of the calling worker thread.
- **packets_received** – **[out]** The vector to fill with packets received.

**struct DummyPacket**
A struct that describes the unit of transmission in the dummy backend (The *DummyPacket*).

A ***JobSlice*** is divided by the worker thread to multiple **DummyPacket* structs which then get sent then received using the backend.

#### Public Members

uint64_t **pkt_id**
A packet identifier unique only within a job slice. Accessed only by the worker thread that created the message. Can be calculated as packet offset from the job slice divided by the packet size

*JobId* **job_id**
The identifier of the job from which this message came from

*Numel* **numel**
The number of elements in the packet

*DataType* **data_type**
The data type of the elements

void ***entries_ptr**
Pointer to data that is supposed to be outstanding (in the network)

void ***extra_info_ptr**
Pointer to extra info that is supposed to be outstanding (in the network)

### Class DummyWorkerThread

- Defined in file_dev_root_client_lib_src_backends_dummy_dummy_worker_thread.h

### Class Documentation

**class** switchml::**DummyWorkerThread**
A class that represents a single dummy worker thread.

A worker thread constantly asks the context for work and carries it out.

Multiple instances of this class is typically created depending on the number of cores in the configuration.

#### Public Functions

**DummyWorkerThread**(*Context* &*context*, *DummyBackend* &*backend*, *Config* &*config*)
Construct a new Dummy Worker Thread object.

**Parameters**

- **context** – **[in]** a reference to the switchml context.
- **backend** – **[in]** a reference to the created dummy backend.
- **config** – **[in]** a reference to the context configuration.

**~DummyWorkerThread**()

**DummyWorkerThread**(*DummyWorkerThread* **const**&) = default

void **operator=**(*DummyWorkerThread* **const**&) = delete

**DummyWorkerThread**(*DummyWorkerThread*&&) = default

*DummyWorkerThread* &**operator=**(*DummyWorkerThread*&&) = default

void **operator()**()
This is the point of entry function for the thread.

void **Start**()
Start the thread.

void **Join**()
Wait for the thread to exit and delete its system reference.

#### Public Members

**const** *WorkerTid* **tid_**
Worker thread id

### Class FifoScheduler

- Defined in file_dev_root_client_lib_src_schedulers_fifo_scheduler.h

### Inheritance Relationships

### Base Type

- `public switchml::Scheduler` (*Class Scheduler*)

### Class Documentation

**class** `switchml::`**FifoScheduler** : **public** switchml::*Scheduler*
A subclass of *Scheduler* that uses a single FIFO queue to store and dispatch jobs.

Jobs are divided into almost-equally-sized job slices where each worker thread works on a single job slice.

This *FifoScheduler* uses a static mapping between the job slices and the worker threads. That means each worker thread will get a known slice of each job and will not compete for slices. For example: If we had 3 worker threads and a job J where J.numel=24 then worker thread 0 will ALWAYS get a slice that includes elements 0-7, thread 1 will ALWAYS get a slice including elements 8-15, thread 3 will ALWAYS get a slice including 16-23 The static mapping is done to avoid collisions at the switch because each worker thread is assigned a unique slot in the switch (at least with the current p4 program version). And we want to make sure that for example elements 0-7 in worker node 0 and worker node 1 are all heading to the same slot in the switch.

#### Public Functions

**FifoScheduler**(*Config* &*config*)
Initialize all the members.

**Parameters** **config** – **[in]** the switchml configuration.

**~FifoScheduler**() = default

**FifoScheduler**(*FifoScheduler* **const**&) = delete

void **operator=**(*FifoScheduler* **const**&) = delete

**FifoScheduler**(*FifoScheduler*&&) = default

*FifoScheduler* &**operator=**(*FifoScheduler*&&) = default

**virtual** bool **EnqueueJob**(std::shared_ptr<*Job*> *job*) **override**
Add a job to the *Scheduler*'s queue.

This function is called by the context after a user submits a new communication job.

**Parameters** **job** – **[in]** a shared pointer for the job that we will enqueue

**Returns** true if we could add the request successfully.

**Returns** false otherwise.

**virtual** bool **GetJobSlice**(*WorkerTid worker_thread_id*, *JobSlice &job_slice*) **override**
Get a job request slice.

This is called through the context by worker threads to get a job slice. How the *Job* is sliced and distributed depends on the scheduler implementation. The function should block the calling thread until a job slice is retrieved.

This function implements a worker thread barrier that ensures that no worker thread gets ahead of other worker threads and that all worker threads are working on the same job. This is unecessary but it allows us to use a single simple queue with constant GetJobSlice time.

**Parameters**

- **worker_thread_id** – **[in]** The id of the worker thread that wants a job slice.
- **job_slice** – **[out]** A reference to a job slice variable.

**Returns** true if the scheduler returned a valid job slice.

**Returns** false the caller was forced to wakeup and the scheduler did not return a valid job slice.

**virtual** bool **NotifyJobSliceCompletion**(*WorkerTid worker_thread_i*, **const** *JobSlice &job_slice*) **override**
Signal the scheduler that a job slice has been finished.

**Parameters**

- **worker_thread_id** – **[in]** The id of the worker thread that finished the job slice.
- **job_slice** – **[in]** The job slice that finished.

**Returns** true If the job corresponding to this job slice has finished all its job slices.

**Returns** false If there is still some job slices to be completed either by other worker threads.

**virtual** void **Stop**() **override**
calls *Scheduler::Stop()*, wakes up all threads waiting, and clears all queues.

After calling the super function *Scheduler::Stop()*, the functions destroys the barrier waking up all threads that are waiting on the barrier. Then the function sets all unfinished jobs to failed thus waking up any threads waiting on a specific job. Finally, it clears queue_, undispatched_job_slices_, and undispatched_job_slices_

### Class GrpcClient

- Defined in file_dev_root_client_lib_src_grpc_client.h

### Class Documentation

**class** switchml::**GrpcClient**

The GRPC client is the mediator between the client library and the controller program.

It can ask the controller to setup switch registers appropriately, and perform simple collective communication operations across all workers (Currently only a barrier and a single value broadcast).

#### Public Functions

**GrpcClient**(*Config* &*config*)

Create stubs and the grpc channel.

**Parameters** **config** – **[in]** a reference to the switchml configuration.

**~GrpcClient**() = default

**GrpcClient**(*GrpcClient* **const**&) = delete

void **operator=**(*GrpcClient* **const**&) = delete

**GrpcClient**(*GrpcClient*&&) = default

*GrpcClient* &**operator=**(*GrpcClient*&&) = default

void **Barrier**(**const** switchml_proto::BarrierRequest &*request*, switchml_proto::BarrierResponse **response*)

A barrier across workers.

**Parameters**

- **request** – **[in]** BarrierRequest containing the number of workers.
- **response** – **[out]** The empty BarrierResponse from the switch.

void **Broadcast**(**const** switchml_proto::BroadcastRequest &*request*, switchml_proto::BroadcastResponse **response*)

Broadcast a value to all workers through the controller.

**Parameters**

- **request** – **[in]**
- **response** – **[out]**

void **CreateRdmaSession**(**const** switchml_proto::RdmaSessionRequest &*request*, switchml_proto::RdmaSessionResponse **response*)

Tell the controller to setup the switch registers for RDMA operation.

**Parameters**

- **request** – **[in]** RdmaSessionRequest containing configuration, session info, memory region info
- **response** – **[out]** RdmaSessionResponse containing the switch's memory region info

void **CreateUdpSession**(**const** switchml_proto::UdpSessionRequest &*request*, switchml_proto::UdpSessionResponse **response*)

Tell the controller to setup the switch registers for UDP operation.

**Parameters**

- **request** – **[in]** UdpSessionRequest containing configuration, session info
- **response** – **[out]** UdpSessionResponse

## Class Job

- Defined in file_dev_root_client_lib_src_job.h

## Class Documentation

**class** switchml::**Job**

A *Job* is used to represent work to be done by SwitchML.

It is created by the *Context* when an operation is requested, submitted to the *Scheduler*, then the scheduler creates instances of *JobSlice* from it to give it to the worker threads.

### Public Functions

**Job**(*Tensor* *tensor*, *JobType* *job_type*, *ExtraJobInfo* *extra_job_info*)

Construct a new *Job* object.

**Parameters**

- **tensor** – **[in]** The tensor to work on for this job.
- **job_type** – **[in]** The type of the job.
- **extra_job_info** – **[in]** Extra information that might be needed for the job.

**~Job**() = default

**Job**(*Job* **const**&) = delete

void **operator=**(*Job* **const**&) = delete

**Job**(*Job*&&) = default

*Job* &**operator=**(*Job*&&) = default

void **WaitToComplete**()

Block the calling thread until the job completes or fails.

*JobStatus* **GetJobStatus**()

Get the job's status.

**Returns** JobStatus

void **SetJobStatus**(*JobStatus* *job_status*)

Update the job's status and notify waiting threads if needed.

This function must only be called by the scheduler or the context. JobStatus must progress in an increasing order.

**Parameters** **job_status** – **[in]** the new job_status

#### Public Members

**const** *JobId* **id_**
Unique identifier for the job.

**const** *Tensor* **tensor_**
*Tensor* to perform the collective communication job on.

**const** *JobType* **job_type_**
The type of collective communication that the job will do.

**const** *ExtraJobInfo* **extra_job_info_**
Extra information specific to the collective communication job.

### Class PrePostProcessor

- Defined in file_dev_root_client_lib_src_prepostprocessor.h

### Inheritance Relationships

### Derived Types

- `public switchml::BypassPPP` (*Class BypassPPP*)
- `public switchml::CpuExponentQuantizerPPP` (*Class CpuExponentQuantizerPPP*)

### Class Documentation

**class** `switchml::`**PrePostProcessor**
A *PrePostProcessor* (PPP) is an object that handles loading and unloading of the data between the client and the network.

Depending on the implementation, the PPP may convert the representation of the data (maybe even compress it) and may require extra information or metadata to be sent so that it can undo its representation changes.

In the prepostprocessor we use 'LTU' to refer to the logical unit of transmission that the backend will use. But the prepostprocessor itself does not care what that "logical transmission unit" really is. Its just dealing with a series of blocks of data that is being sent and received. Call it a packet (for dpdk), a block, a message (in rdma).

Subclassed by *switchml::BypassPPP*, *switchml::CpuExponentQuantizerPPP*

#### Public Functions

**~PrePostProcessor**() = default

**PrePostProcessor**(*PrePostProcessor* **const**&) = delete

void **operator=**(*PrePostProcessor* **const**&) = delete

**PrePostProcessor**(*PrePostProcessor*&&) = default

*PrePostProcessor* &**operator=**(*PrePostProcessor*&&) = default

**virtual** uint64_t **SetupJobSlice**(*JobSlice* **job_slice*) = 0
Setup the PPP's internal structures and prepare to start processing the passed job slice.

**Parameters** **job_slice** – **[in]** A pointer to the job slice currently being worked on by the worker thread.

**Returns** uint64_t the number of transmission units that prepostprocessor will need to be sent and received by the backend so that the whole tensor is processed. (This does not include LTUs from the extra batch that might be needed). We let the PPP return this information so that the backend is aware in case the PPP reduces the size of the data and thus needs a smaller number of LTUs to be transmitted.

**virtual** bool **NeedsExtraBatch**() = 0

Check whether this prepostprocessor needs to send an extra batch for the current job slice or not.

Some prepostprocessor's need extra info / metadata to be sent along the payload so that they convert between representations correctly. And they usually need that extra info to be present before the first real batch is sent. In that case the backend sends an extra first batch to make this information available for the first real batch later.

**Returns** true If the prepostprocessor needs an extra batch

**Returns** false If it doesn't

**virtual** void **PreprocessSingle**(uint64_t *ltu_id*, void **entries_ptr*, void **extra_info* = nullptr) = 0

Preprocess an LTU converting its representation if needed and moving its payload into the backend's buffers.

**See** *PostprocessSingle()*

**Parameters**

- **ltu_id** – **[in]** The id of the logical transmission unit to be preprocessed within the current job slice. ltu_id will be used to compute the offset into the job slice ltu_id * ltu_size.
- **entries_ptr** – **[out]** A pointer to where we will store the payload to be ready for transmission.
- **extra_info** – **[out]** A pointer to where we will store the extra info if we need it.

**virtual** void **PostprocessSingle**(uint64_t *ltu_id*, void **entries_ptr*, void **extra_info* = nullptr) = 0

Postprocess an LTU converting it to the original representation if needed and moving its payload into the client's buffers.

**See** *PreprocessSingle()*

**Parameters**

- **ltu_id** – **[in]** The id of the logical transmission unit to be postprocessed within the current job slice. ltu_id will be used to compute the offset into the job slice ltu_id * ltu_size.
- **entries_ptr** – **[in]** A pointer to where we will read the received payload from.
- **extra_info** – **[in]** A pointer to where we will read the extra info from if we need it.

**virtual** void **CleanupJobSlice**() = 0

Cleans up all internal structures and released any dynamically allocated memory associated with the job slice.

**See** *SetupJobSlice()*

### Public Static Functions

**static** std::shared_ptr<*PrePostProcessor*> **CreateInstance**(*Config* &*config*, *WorkerTid* *worker_tid*, *Numel* *ltu_size*, *Numel* *batch_num_ltus*)

Create an instance of the prepostprocessor specified in the configuration passed.

**Parameters**

- **config** – **[in]** A reference to the context's configuration.
- **worker_tid** – **[in]** The worker thread that this prepostprocessor belongs to.
- **ltu_size** – **[in]** The size in bytes of the logical transmission unit used by the backend.
- **batch_num_ltus** – **[in]** How many LTUs constitute a batch.

**Returns** std::shared_ptr<PrePostProcessor> a shared pointer to the prepostprocessor's created instance.

### Protected Functions

**PrePostProcessor**(*Config* &*config*, *WorkerTid* *worker_tid*, *Numel* *ltu_size*, *Numel* *batch_num_ltus*)

Construct a new *PrePostProcessor* object.

**Parameters**

- **config** – **[in]** A reference to the context configuration
- **worker_tid** – **[in]** The worker thread that this prepostprocessor belongs to
- **ltu_size** – **[in]** The size in bytes of the logical transmission unit used by the backend.
- **batch_num_ltus** – **[in]** How many LTUs constitute a batch.

### Protected Attributes

*Config* &**config_**

A reference to the context configuration

*WorkerTid* **worker_tid_**

The worker thread that this prepostprocessor belongs to

*Numel* **ltu_size_**

The size in bytes of the logical transmission unit used by the backend.

*Numel* **batch_max_num_ltus_**

What's the maximum number of LTUs that can constitute a batch.

### Class RdmaBackend

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_backend.h

### Inheritance Relationships

### Base Type

- `public switchml::Backend` (*Class Backend*)

### Class Documentation

**class** `switchml::RdmaBackend` : **public** switchml::*Backend*

The backend that represents the rdma version of switchml.

#### Public Functions

**RdmaBackend**(*Context* &*context*, *Config* &*config*)

Call the super class constructor.

**Parameters**

- **context** – **[in]** The context
- **config** – **[in]** The context configuration.

**~RdmaBackend**()

**RdmaBackend**(*RdmaBackend* **const**&) = delete

void **operator=**(*RdmaBackend* **const**&) = delete

**RdmaBackend**(*RdmaBackend*&&) = default

*RdmaBackend* &**operator=**(*RdmaBackend*&&) = default

**virtual** void **SetupWorker**() **override**

Establish the RDMA connection, setup the switch, and start the worker threads.

**virtual** void **CleanupWorker**() **override**

Wait for all worker threads to exit.

std::unique_ptr<*RdmaConnection*> &**GetConnection**()

Get the RDMA connection object that the worker threads will use to send and receive.

**Returns** std::unique_ptr<RdmaConnection>&

### Class RdmaConnection

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_connection.h

### Class Documentation

**class** switchml::**RdmaConnection**

The *RdmaConnection* represents the connection to both the controller and the switch.

It is used by the backend to setup the connection by exchanging the needed information with the controller via the *GrpcClient*, sets up and brings up queue pairs, and finally worker threads then use it to send and receive messages.

#### Public Functions

**RdmaConnection**(*Config* &*config*)

Initialize all members and allocate buffer memory region.

**Parameters** **config** – **[in]** a reference to the switchml configuration.

**~RdmaConnection**()

**RdmaConnection**(*RdmaConnection* **const**&) = delete

void **operator=**(*RdmaConnection* **const**&) = delete

**RdmaConnection**(*RdmaConnection*&&) = default

*RdmaConnection* &**operator=**(*RdmaConnection*&&) = default

void **Connect**()

Performs all needed setup and bringup to establish the RDMA connection.

This should be the first function to be called after creating the object. After calling this function, you can go ahead and use the getters to access the created queue pairs, memory region and so on. You can also then use the *PostSend()* and *PostRecv()* functions to send and receive messages.

ibv_cq *__GetWorkerThreadCompletionQueue__(*WorkerTid worker_thread_id*)

std::vector<ibv_qp*> **GetWorkerThreadQueuePairs**(*WorkerTid worker_thread_id*)

Get the range of queue pairs corresponding to a worker thread.

**Parameters** **worker_thread_id** – **[in]**

**Returns** std::vector<ibv_qp*>

std::vector<uint32_t> **GetWorkerThreadRkeys**(*WorkerTid worker_thread_id*)

Get the range of rkeys corresponding to a worker thread.

**Parameters** **worker_thread_id** –

**Returns** std::vector<uint32_t>

std::pair<void*, uint32_t> **GetWorkerThreadMemoryRegion**(*WorkerTid worker_thread_id*)

Get the memory region information corresponding to a worker thread.

**Parameters** **worker_thread_id** –

**Returns** std::pair<void*, uint32_t> first element is the first address in the memory region that the worker thread can access. Second element is the lkey of the memory region.

*RdmaEndpoint* &**GetEndpoint**()
Get the underlying used endpoint.

**Returns** *RdmaEndpoint*&

### Public Static Functions

**static** void **PostRecv**(ibv_qp *_qp_, ibv_recv_wr *_wr_)
Post receive work request and check its success.

**Parameters**

- **qp** – **[in]** The queue pair to use.
- **wr** – **[in]** The receive work request to post

**static** void **PostSend**(ibv_qp *_qp_, ibv_send_wr *_wr_)
Post send work request and check its success.

**Parameters**

- **qp** – **[in]** The queue pair to use
- **wr** – **[in]** The send work request to post.

## Class RdmaEndpoint

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_endpoint.h

## Class Documentation

**class** switchml::**RdmaEndpoint**
The *RdmaEndpoint* class contains all functions and configurations to setup the machine and the device/NIC.

It is mainly used by the *RdmaConnection* class.

### Public Functions

**RdmaEndpoint**(std::string *_device_name_*, uint16_t *_device_port_id_*, uint16_t *_gid_index_*)
Initialize members and configure and open the ibverbs device port.

**Parameters**

- **device_name** – The name of the Infiniband device to use.
- **device_port_id** – The specific port to use from the device.
- **gid_index** – The GID index to use.

**~RdmaEndpoint**()

**RdmaEndpoint**(*RdmaEndpoint* **const**&) = delete

void **operator=**(*RdmaEndpoint* **const**&) = delete

**RdmaEndpoint**(*RdmaEndpoint*&&) = default

*RdmaEndpoint* &**operator=**(*RdmaEndpoint*&&) = default

ibv_mr *`AllocateAtAddress`(void *_requested_address_, uint64_t _size_)
Allocates and registers a memory region at a specific address.

The call will fail if the allocation is not possible.

**Parameters**

- **`requested_address`** – The memory address wanted.
- **`size`** – The size of the region in bytes.

**Returns** ibv_mr* A pointer to the allocated memory region struct.

void `free`(ibv_mr *_mr_)
Free a memory region that was allocated and registered.

**Parameters** **`mr`** – The memory region to free.

ibv_cq *`CreateCompletionQueue`()
Create a Completion Queue.

**Returns** ibv_cq* The created completion queue.

ibv_qp *`CreateQueuePair`(ibv_cq *_completion_queue_)
Create a Queue Pair.

**Parameters** **`completion_queue`** – The completion queue to associate with this queue pair.

**Returns** ibv_qp* The created queue pair.

uint64_t `GetMac`()
Get the MAC address corresponding to the chosen GID.

**Returns** uint64_t The MAC address.

uint32_t `GetIPv4`()
Get the IPv4 address corresponding to the chosen GID.

**Returns** uint32_t The IPv4 address

ibv_port_attr `GetPortAttributes`()

ibv_device *`GetDevice`()

### Class RdmaWorkerThread

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_worker_thread.h

### Class Documentation

**`class`** `switchml::RdmaWorkerThread`
A class that represents a single rdma worker thread.

A worker thread constantly asks the context for work and carries it out.

Multiple instances of this class is typically created depending on the number of threads in the configuration.

### Public Functions

**RdmaWorkerThread**(*Context* &*context*, *RdmaBackend* &*backend*, *Config* &*config*)
Construct a new RDMA Worker Thread object.

**Parameters**

- **context** – **[in]** a reference to the switchml context.
- **backend** – **[in]** a reference to the created rdma backend.
- **config** – **[in]** a reference to the context configuration.

**~RdmaWorkerThread**()

**RdmaWorkerThread**(*RdmaWorkerThread* **const**&) = default

void **operator=**(*RdmaWorkerThread* **const**&) = delete

**RdmaWorkerThread**(*RdmaWorkerThread*&&) = default

*RdmaWorkerThread* &**operator=**(*RdmaWorkerThread*&&) = default

void **operator()**()
This is the point of entry function for the thread.

void **Start**()
Start the thread.

void **Join**()
Wait for the thread to exit and delete its system reference.

### Public Members

**const** *WorkerTid* **tid_**
Worker thread id

## Class Scheduler

- Defined in file_dev_root_client_lib_src_scheduler.h

## Inheritance Relationships

## Derived Type

- `public switchml::FifoScheduler` (*Class FifoScheduler*)

### Class Documentation

**class** switchml::**Scheduler**

The scheduler class which is responsible for distributing jobs across worker threads.

The scheduler implementation can choose any algorithm, queuing design, data structure, or distribution mechanism to serve its purpose.

The scheduler should only be accessed through the context api. Any scheduler implementation must be thread safe in the sense that it locks the scheduler access lock before any function and releases it before exiting.

If more fine grained locking is needed then the implementation can create its own locks.

Subclassed by *switchml::FifoScheduler*

#### Public Functions

**~Scheduler**() = default

**Scheduler**(*Scheduler* **const**&) = delete

void **operator=**(*Scheduler* **const**&) = delete

**Scheduler**(*Scheduler*&&) = default

*Scheduler* &**operator=**(*Scheduler*&&) = default

**virtual** bool **EnqueueJob**(std::shared_ptr<*Job*> *job*) = 0

Add a job to the *Scheduler*'s queue.

This function is called by the context after a user submits a new communication job.

**Parameters** **job** – **[in]** a shared pointer for the job that we will enqueue

**Returns** true if we could add the request successfully.

**Returns** false otherwise.

**virtual** bool **GetJobSlice**(*WorkerTid worker_thread_id*, *JobSlice* &*job_slice*) = 0

Get a job request slice.

This is called through the context by worker threads to get a job slice. How the *Job* is sliced and distributed depends on the scheduler implementation. the function will block the calling thread on the job_submitted_event_ until a job slice is retrieved OR the Stop is called. This is why it is important to check for the return value to make sure that a job slice has been received.

**Parameters**

- **worker_thread_id** – **[in]** The id of the worker thread that wants a job slice.
- **job_slice** – **[out]** A reference to a job slice variable.

**Returns** true if the scheduler returned a valid job slice.

**Returns** false the caller was forced to wakeup and the scheduler did not return a valid job slice.

**virtual** bool **NotifyJobSliceCompletion**(*WorkerTid worker_thread_id*, **const** *JobSlice* &*job_slice*) = 0

Signal the scheduler that a job slice has been finished.

Since the scheduler is the one responsible for creating job slices out of jobs, it is the only entity that can know when a job is completed.

**Parameters**

- **worker_thread_id** – **[in]** The id of the worker thread that finished the job slice.
- **job_slice** – **[in]** The job slice that finished.

**Returns** true If the job corresponding to this job slice has been fully completed.

**Returns** false If there is still some job slices to be completed either by the calling worker thread or others.

**virtual** void **Stop**()
Set the stopped_ flag to true and notify threads waiting on the job submitted event.

Each implementation should work to wakeup any waiting threads whether waiting on jobs or the scheduler itself. It should also clear any dynamically allocated state.

### Public Static Functions

**static** std::unique_ptr<*Scheduler*> **CreateInstance**(*Config* &*config*)
Creates a *Scheduler* object based on the scheduler name in the config.

**Parameters** **config** – a reference to the context configuration.

**Returns** std::unique_ptr<Scheduler> an exclusive pointer to the created scheduler object.

### Protected Functions

**Scheduler**(*Config* &*config*)
Construct a new *Scheduler* object.

**Parameters** **config** – **[in]** A reference to the context configuration

### Protected Attributes

bool **stopped_**
A flag that signifies that the scheduler has been stopped_

std::mutex **access_mutex_**
A mutex that is used to wrap all functions of the scheduler to make them thread safe.

std::condition_variable **job_submitted_event_**
A condition variable used by GetJobSlice to block until a job is available.

*Config* &**config_**
A reference to the context configuration

## Class Stats

- Defined in file_dev_root_client_lib_src_stats.h

### Class Documentation

**class** switchml::**Stats**

A class that groups up all statistics.

The class does no attempt to syncrhonize in any of its calls. It is the user classes responsibility to synchronize when needed.

#### Public Functions

**Stats**()

Initialize all members.

*InitStats()* must be called before any of the stats functions are used.

**See** *InitStats()*

**~Stats**()

Cleans up the memory that has been allocated by *InitStats()*

**See** *InitStats()*

**Stats**(*Stats* **const**&) = delete

void **operator=**(*Stats* **const**&) = delete

**Stats**(*Stats*&&) = default

*Stats* &**operator=**(*Stats*&&) = default

void **InitStats**(*WorkerTid num_worker_threads*)

Dynamically allocate necessary objects and reset all stats using *ResetStats()*.

**Parameters** **num_worker_threads** – **[in]** The number of worker threads which will use this stats object.

void **LogStats**()

Parse and log all of the statistics using glog.

void **ResetStats**()

Clear all accumulated statistics.

std::string **DescribeIntList**(std::vector<uint64_t> *list*)

Describe the distribution of a list of integers.

Computes sum, mean, max, min, median, stdev

**Parameters** **list** – The list of integers to describe

**Returns** std::string a single line with all of the metrics.

std::string **DescribeFloatList**(std::vector<double> *list*)

Describe the distribution of a list of doubles.

Computes sum, mean, max, min, median, stdev

**Parameters** **list** – The list of doubles to describe

**Returns** std::string a single line with all of the metrics.

template<typename **T**>

std::string **List2Str**(std::vector<*T*> *list*)
Create a string representation of a list.

**Parameters** **list** – The vector representing the list

**Returns** std::string A single line with all of the elements

**inline** void **IncJobsSubmittedNum**()

**inline** void **AppendJobSubmittedNumel**(uint64_t *size*)

**inline** void **IncJobsFinishedNum**()

**inline** void **AddTotalPktsSent**(*WorkerTid* *wtid*, uint64_t *to_add*)

**inline** void **AddCorrectPktsReceived**(*WorkerTid* *wtid*, uint64_t *to_add*)

**inline** void **AddWrongPktsReceived**(*WorkerTid* *wtid*, uint64_t *to_add*)

**inline** void **AddTimeouts**(*WorkerTid* *wtid*, uint64_t *to_add*)

### Class TimeoutQueue

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_timeout_queue.h

### Nested Relationships

### Nested Types

- *Struct TimeoutQueue::TQEntry*

### Class Documentation

**class** switchml::**TimeoutQueue**
An efficient data structure used to check for message timeouts.

In order to ensure that all operations are done in constant time, the timeoutqueue was designed using an ordered double-linked list which also has an index for entries. This allows us to perform all of the 3 functions (push, remove, check) in constant time.

#### Public Types

**using** **TimePoint** = std::chrono::time_point<switchml::*clock*>
Type of timestamp. Just an alias for user convenience.

### Public Functions

**TimeoutQueue**(**const** uint32_t *num_entries*, **const** std::chrono::milliseconds *timeout*, **const** uint32_t *timeouts_threshold*, **const** uint32_t *timeouts_threshold_increment*)
Construct a new Timeout Queue object.

**Parameters**

- **num_entries** – **[in]** The maximum number of entries that you might push. This should equal the number of outstanding messages.
- **timeout** – **[in]** The initial value of the timeout in milliseconds.
- **threshold** – **[in]** After how many timeouts should we double the timeout value?.
- **threshold** – **[in]** After a timeout occurs how much should we increment the threshold?.

**~TimeoutQueue**() = default

**TimeoutQueue**(*TimeoutQueue* **const**&) = default

void **operator=**(*TimeoutQueue* **const**&) = delete

**TimeoutQueue**(*TimeoutQueue*&&) = default

*TimeoutQueue* &**operator=**(*TimeoutQueue*&&) = default

void **Push**(int *index*, **const** *TimePoint* &*timestamp*)
Push an entry onto the queue.

Elements are added to the top of the linked list because they are always assumed to be the newest. This allows us to keep the order and insert into the linked list in constant time.

**Parameters**

- **index** – **[in]** The index where you want to store the entry for direct access later. This is not the same as the entry's position in the linked list.
- **timestamp** – **[in]** The current timestamp

void **Remove**(int *index*)
Remove an entry.

This operation is done in constant time since the index gives us direct access to the linked list element and we only then need to rewire the pointers of the two adjacent entries in the linked list if they exist.

**Parameters** **index** – **[in]** The index of the entry to remove.

int **Check**(**const** *TimePoint* &*timestamp*)
Given the current timestamp, check a timeout occured.

If a timeout occured, then the index of the entry that timed out first is returned.

This is also done in constant time since we only need to check the entry that exists at the tail of the linked list.

**Parameters** **timestamp** – **[in]** The current timestamp.

**Returns** int the index of the entry that timed out first.

**struct TQEntry**
A struct representing a single entry in the *TimeoutQueue*.

#### Public Functions

**TQEntry**()

**~TQEntry**() = default

**TQEntry**(*TQEntry* **const**&) = default

void **operator=**(*TQEntry* **const**&) = delete

**TQEntry**(*TQEntry*&&) = default

*TQEntry* &**operator=**(*TQEntry*&&) = default

#### Public Members

bool **valid**
: Whether this entry is just a place holder or an actual entry pushed by the user.

int **next**
: The index of the next entry

int **previous**
: The index of the previous entry

*TimePoint* **timestamp**
: The time at which this entry was pushed

### 3.3.3 Enums

#### Enum AllReduceOperation

- Defined in file_dev_root_client_lib_src_job.h

#### Enum Documentation

**enum** switchml::**AllReduceOperation**
: The operation to use when performing AllReduce.

*Values:*

**enumerator SUM**
: Use summation to reduce the tensors

#### Enum DataType

- Defined in file_dev_root_client_lib_src_common.h

### Enum Documentation

**enum** switchml::**DataType**
Numerical data type enum.

*Values:*

**enumerator FLOAT32**
Represents a standard float type

**enumerator INT32**
Represents a standard 32 bit signed integer

### Enum JobStatus

- Defined in file_dev_root_client_lib_src_job.h

### Enum Documentation

**enum** switchml::**JobStatus**
Describes the current status of a *Job* instance.

*Values:*

**enumerator INIT**
The job was just created.

**enumerator QUEUED**
The job has been added to the scheduler's queue.

**enumerator RUNNING**
Some worker threads are currently working on slices of the job.

**enumerator FINISHED**
All job slices have been completed and the job finished successfully.

**enumerator FAILED**
The job failed for some reason.

### Enum JobType

- Defined in file_dev_root_client_lib_src_job.h

### Enum Documentation

**enum** switchml::**JobType**
The type of collective communication job.

*Values:*

**enumerator ALLREDUCE**
Perform an AllReduce operation

**enumerator BROADCAST**
Perform a Broadcast operation. **Not yet supported**

### 3.3.4 Unions

#### Union ExtraJobInfo

- Defined in file_dev_root_client_lib_src_job.h

#### Union Documentation

**union** switchml::**ExtraJobInfo**
*#include <job.h>* Extra information specific to the collective communication job.

##### Public Members

*AllReduceOperation* **allreduce_operation**
The operation to use for AllReduce

int32_t **broadcast_root_rank**
The worker that is broadcasting so it knows that it should send and others will receive.

### 3.3.5 Functions

#### Function switchml::BindToCore

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_utils.h

#### Function Documentation

**inline** void switchml::**BindToCore**(ibv_device ***device*, uint32_t *worker_id*)
A function to bind the calling thread to an appropriate core.

**Parameters** **worker_id** –

#### Function switchml::ChangeMacEndianness

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_utils.h

#### Function Documentation

uint64_t switchml::**ChangeMacEndianness**(uint64_t *mac*)
Take a MAC address as an 8 bytes integer with the first 6 bytes representing the MAC address and convert its endianness.

**Parameters** **mac** – an 8 bytes integer with the first 6 bytes representing the original MAC address.

**Returns** uint64_t an 8 bytes integer with the first 6 bytes representing the converted MAC address.

### Function switchml::DataTypeSize

- Defined in file_dev_root_client_lib_src_common.h

### Function Documentation

**static inline** uint16_t switchml::**DataTypeSize**(**enum** *DataType type*)
Returns the size of an element of the given DataType.

**Parameters** **type** – **[in]** the data type to ask about.

**Returns** uint16_t The size of a an element of this data type.

### Function switchml::Execute

- Defined in file_dev_root_client_lib_src_utils.h

### Function Documentation

**inline** std::string switchml::**Execute**(**const** char **cmd*)
A function to execute any command on the system and return the standard output as a string.

**Parameters** **cmd** – **[in]** the command to execute.

**Returns** std::string a string that contains the standard output of the command.

### Function switchml::GetCoresNuma

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_utils.h

### Function Documentation

**inline** std::unordered_map<int, std::vector<int>> switchml::**GetCoresNuma**()
A function to query the system and get all Core ids grouped up by their NUMA nodes.

**Returns** std::unordered_map<int, std::vector<int>> A map with the NUMA node as the key and a vector of physical core ids that reside on that numa node.

### Function switchml::GetDeviceNuma

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_utils.h

### Function Documentation

**inline** int switchml::**GetDeviceNuma**(ibv_device **device*)
Query the system to find the NUMA node on which the given device resides.

**Parameters** **device** – **[in]**

**Returns** int the NUMA node which this ibverbs device resides on.

## Function switchml::GIDToIPv4

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_utils.h

### Function Documentation

**inline** uint32_t switchml::**GIDToIPv4**(**const** ibv_gid *gid*)
Extract the IPv4 address from a GID address.

**See** *IPv4ToGID()*

**Parameters** **gid** – **[in]** GID to extract from.

**Returns** uint32_t The IP address as a 32 bit integer.

## Function switchml::GIDToMAC

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_utils.h

### Function Documentation

**inline** uint64_t switchml::**GIDToMAC**(**const** ibv_gid *gid*)
Extract the MAC address from a GID address.

**See** *MACToGID()*

**Parameters** **gid** – **[in]** GID to extract from.

**Returns** uint64_t The MAC address as a 64 bit integer (The two most significant bytes are ignored).

## Function switchml::IPv4ToGID

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_utils.h

### Function Documentation

**inline** ibv_gid switchml::**IPv4ToGID**(**const** int32_t *ip*)

Create GID from IPv4 address.

**See** *GIDToIPv4()*

**Parameters** **ip** – **[in]** IP address as a 32 bit integer.

**Returns** ibv_gid The created GID address.

### Function switchml::Mac2Str(const uint64_t)

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_utils.h

### Function Documentation

std::string switchml::**Mac2Str**(**const** uint64_t *addr*)

Take a MAC address as an 8 bytes integer with the first 6 bytes representing the MAC address and return the string representation of it.

**Parameters** **addr** – an 8 bytes integer with the first 6 bytes representing the MAC address.

**Returns** std::string the string represntation of the MAC address (FF:FF:FF:FF:FF:FF)

### Function switchml::Mac2Str(const rte_ether_addr)

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_utils.h

### Function Documentation

std::string switchml::**Mac2Str**(**const** rte_ether_addr *addr*)

Take a MAC address as an array of 6 bytes and return the string representation of it.

**Parameters** **addr** – 6 byte array representing the MAC address

**Returns** std::string the string represntation of the MAC address (FF:FF:FF:FF:FF:FF)

### Function switchml::MACToGID

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_utils.h

#### Function Documentation

**inline** ibv_gid switchml::**MACToGID**(**const** uint64_t *mac*)
Create GID from MAC address.

**See** *GIDToMAC()*

**Parameters** **mac** – **[in]** Mac address as a 64 bit integer (The two most significant bytes are ignored).

**Returns** ibv_gid The created GID address.

### Function switchml::Str2Mac

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_utils.h

#### Function Documentation

uint64_t switchml::**Str2Mac**(std::string **const** &*mac_str*)
Take a string representation of the MAC address and return 8 bytes integer. with the first 6 bytes representing the MAC address.

**Parameters** **mac_str** – the string represntation of the MAC address (FF:FF:FF:FF:FF:FF)

**Returns** an 8 bytes integer with the first 6 bytes representing the MAC address.

### Template Function switchml::ToHex

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_utils.h

#### Function Documentation

template<typename **T**>
std::string switchml::**ToHex**(*T*)
Takes a data element and returns the hex string that represents its bits.

**Returns** std::string the hex string of the bytes stored in the passed data element

## 3.3.6 Variables

### Variable job_type_size

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_backend.h

### Variable Documentation

uint8_t switchml::*DpdkBackend*::*DpdkPacketHdr*::**job_type_size**
: This field is used to store both the job type and the packet's size enum or category. The 4 MSBs are for the job type and the 4 LSBs are for the size.

### Variable pkt_id

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_backend.h

### Variable Documentation

uint32_t switchml::*DpdkBackend*::*DpdkPacketHdr*::**pkt_id**
: An id to identify a packet within a job slice. This is used by the client only.

### Variable short_job_id

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_backend.h

### Variable Documentation

uint8_t switchml::*DpdkBackend*::*DpdkPacketHdr*::**short_job_id**
: The 8 LSBs of the id of the job associated with this packet. This is used by the client only to discard duplicates at the edge of switching from one job to another. Therefore we do not need the full length of the job id.

### Variable switch_pool_index

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_backend.h

### Variable Documentation

uint16_t switchml::*DpdkBackend*::*DpdkPacketHdr*::**switch_pool_index**
: The switch's pool/slot index.

  A pool or a slot in the switch is what is used to store the values of a packet. Think of the switch as a large array of pools or slots. Each packet sent addresses a particular pool/slot.

  The MSB of this field is used to alternate between two sets of pools/slots to have a shadow copy for switch retransmissions.

### 3.3.7 Defines

#### Define DPDK_SWITCH_ELEMENT_SIZE

- Defined in file_dev_root_client_lib_src_backends_dpdk_dpdk_backend.h

#### Define Documentation

**DPDK_SWITCH_ELEMENT_SIZE**

#### Define DUMMY

- Defined in file_dev_root_client_lib_src_backends_dummy_dummy_backend.h

#### Define Documentation

**DUMMY**

#### Define DUMMY_ELEMENT_SIZE

- Defined in file_dev_root_client_lib_src_backends_dummy_dummy_backend.h

#### Define Documentation

**DUMMY_ELEMENT_SIZE**

#### Define RDMA_SWITCH_ELEMENT_SIZE

- Defined in file_dev_root_client_lib_src_backends_rdma_rdma_backend.h

#### Define Documentation

**RDMA_SWITCH_ELEMENT_SIZE**

### 3.3.8 Typedefs

#### Typedef switchml::clock

- Defined in file_dev_root_client_lib_src_common.h

### Typedef Documentation

**typedef** std::chrono::steady_clock `switchml::`**clock**
The clock type used in all time measurements for switchml.

### Typedef switchml::JobId

- Defined in file_dev_root_client_lib_src_common.h

### Typedef Documentation

**typedef** uint64_t `switchml::`**JobId**
Type used to represent all job ids.

### Typedef switchml::Numel

- Defined in file_dev_root_client_lib_src_common.h

### Typedef Documentation

**typedef** uint64_t `switchml::`**Numel**
Type used to represent the number of elements in all tensors

### Typedef switchml::WorkerTid

- Defined in file_dev_root_client_lib_src_common.h

### Typedef Documentation

**typedef** int16_t `switchml::`**WorkerTid**
Type used to represent all worker thread ids.

CHAPTER

FOUR

# SWITCHML P4 PROGRAM

The SwitchML P4 program is written in P4-16 for the Tofino Native Architecture (TNA) and the controller uses the Barefoot Runtime Interface (BRI) to program the switch.

## 4.1 1. Requirements

The P4 code has been tested on Intel P4 Studio 9.6.0.

For details on how to obtain and compile P4 Studio, we refer you to the official Intel documentation.

The document *[1]* provides all the instructions to compile P4 Studio (aka SDE) and a P4 program. Here we show one possible way to compile the SDE using the P4 Studio build tool.

Assuming that the `SDE` environment variable points to the SDE folder, you can use the following commands to compile it for P4-16 development:

```
cd $SDE/p4studio
sudo -E ./install-p4studio-dependencies.sh
./p4studio profile apply ./profiles/all-tofino.yaml
```

You might also need to compile a BSP package, depending on your switch platform. The Intel documentation has the BSP package and compilation instructions for the reference platform.

The control plane requires python 3.8, so P4 Studio must be compiled using python 3, so that the generated Barefoot Runtime libraries will be compatible with python 3.

## 4.2 2. Running the P4 program

1. Build the P4 code. Detailed instructions are available in the Intel documentation *[1]*. Assuming that you are currently in the p4 directory, one way to compile the P4 program is with the following commands:

   ```
   mkdir build && cd build
   cmake $SDE/p4studio/ -DCMAKE_INSTALL_PREFIX=$SDE_INSTALL \
                        -DCMAKE_MODULE_PATH=$SDE/cmake \
                        -DP4_NAME=SwitchML \
                        -DP4_PATH=`pwd`/../switchml.p4
   ```

2. Run the reference driver application:

   ```
   $SDE/run_switchd.sh -p SwitchML
   ```

3. When switchd is started, run the control plane program (either on a switch or on a separate server).

## 4.3 3. Design

**This section is a work in progress**

### 4.3.1 3.1 SwitchML packet formats:

SwitchML currently supports two packet formats: UDP and RoCEv2.

With UDP, SwitchML packets carry a dedicated header between UDP and the payload. A range of UDP ports [0xBEE0, 0xBEEF] are used as destination/source ports in packets going received/sent by the switch. Currently we support a payload that is either 256B or 1024B (using recirculation). This is the overall packet format:

| Ethernet | IPv4 | UDP | SwitchML | Payload | Ethernet FCS |
|---|---|---|---|---|---|

With RDMA, the packet layout is slightly different depending on which part of a message a packet contains. A message with a single packet looks like this:

| Ethernet | IPv4 | UDP | IB BTH | IB RETH | IB IMM | Payload | IB ICRC | Ethernet FCS |
|---|---|---|---|---|---|---|---|---|

The P4 program does not check nor update the ICRC value, so the end-host servers should disable ICRC checking.

## 4.4 References

[1] Intel® P4 Studio Software Development Environment (SDE) 9.6.0 Installation Guide

CHAPTER

FIVE

# SWITCH CONTROLLER

The SwitchML controller will program the switch at runtime using the Barefoot Runtime Interface (BRI). The controller accepts connections from end-hosts through gRPC to set up a job (which is a sequence of allreduce operations involving the same set of workers). It also provides a CLI interface that can be used to configure the switch and read counters values at runtime.

## 5.1 Requirements

The controller requires python 3.8 and the following python packages:

```
grpcio>=1.34.0 pyyaml asyncio ipaddress ansicolors
```

It also requires the gRPC code autogenerated from the .proto file, which is done by running `make` in the controller folder. If you installed GRPC on your own then you should pass `GRPC_HOME=path_to_grpc_installation` to the make command.

Additionally, the following two modules are required:

```
bfrt_grpc.bfruntime_pb2 bfrt_grpc.client
```

These modules are autogenerated when P4 Studio is compiled. The controller expects that the `SDE_INSTALL` environment variable points to the SDE install directory. It will search for those modules in the following folder:

```
$SDE_INSTALL/lib/python*/site-packages/tofino/bfrt_grpc/
```

## 5.2 Running the controller

To enable switch ports and configure the switch to forward regular traffic, the controller reads the `ports.yml` file that describes the machines connected to the switch ports. Each front panel port is identified with port number and lane. The parameters per port are:

- speed (one of 10G, 25G, 40G, 50G, 100G, the default is 100G)
- fec (one of none, fc, rs, the default is "none")
- autoneg (one of default, enable, disable, the default is "default")
- mac (mac address of the NIC connected to the port)

This is an example:

```
ports:
    1/0 : {speed: "100G", fec: "none", autoneg: "disable", mac: "00:11:22:33:44:55"}
    2/0 : {speed: "100G", fec: "none", autoneg: "disable", mac: "00:11:22:33:44:66"}
```

The controller is started with:

```
python switchml.py
```

The optional arguments are the following:

| Argument | Description | Default |
|---|---|---|
| –program PROGRAM | P4 program name | SwitchML |
| –bfrt-ip ADDRESS | Name/address of the BFRuntime server | 127.0.0.1 |
| –bfrt-port PORT | Port of the BFRuntime server | 50052 |
| –switch-mac SWITCH_MAC | MAC address of the switch | 00:11:22:33:44:55 |
| –switch-ip SWITCH_IP | IP address of switch | 10.0.0.254 |
| –ports FILE | YAML file describing machines connected to ports | ports.yaml |
| –log-level LEVEL | Logging level: ERROR, WARNING, INFO, DEBUG | INFO |

The BFRuntime server is the switch reference drivers application. The switch MAC and IP are the addresses of your choosing that will be used by the switch when acting as a SwitchML endpoint.

CHAPTER

SIX

# EXAMPLES

The examples directory includes multiple simplified well commented examples to show how the different SwitchML parts can be used in different scenarios.

## 6.1 1. Examples list

| Example | Brief |
|---|---|
| hello_world | A minimal example showing how the SwitchML client library can be used through the SwitchML context. |

## 6.2 2. Compiling

All examples require that the client library be compiled and that the SwitchML configuration file is present when running.

Also note that linking the client library code happens here. So you usually need to provide the same build variables that you used when you compiled the client library in addition to the ones that control the example itself. This allows the Makefile to link to the appropriate libraries for DPDK, RDMA, etc.

To compile an example, simply run (Assuming you are in the examples directory):

```
make <example_name> [build variables]
```

Or to compile all examples at once just run

```
make [build variables]
```

By default, the examples executables will be found in

```
dev_root/build/bin/
```

### 6.2.1 2.1 Build variables

The following variables can all be passed to the examples makefile to control the build.

| Variable | Type | Default | Usage |
|---|---|---|---|
| DEBUG | boolean | 0 | Disable optimizations, add debug symbols. |
| DPDK | boolean | 0 | Add dpdk backend specific compiler/linker options. |
| MLX5 | boolean | 0 | Add dpdk backend Connect-x5/Connect-x4 specific compiler/linker options. |
| MLX4 | boolean | 0 | Add dpdk backend Connect-x3 specific compiler/linker options. |
| RDMA | boolean | 0 | Add rdma backend specific compiler/linker options. |
| BUILDDIR | path | dev_root/build | Where to store generated objects/binaries |
| SWITCHML_HOME | path | dev_root/build | Where to look for the switchml client library installation |
| GRPC_HOME | path | dev_root/third_party/grpc/build | Where to look for the GRPC installation |
| DPDK_HOME | path | dev_root/third_party/dpdk | Where to look for the DPDK installation |

CHAPTER

SEVEN

# BENCHMARKS

The benchmarks directory includes multiple benchmarks to test and measure the performance of the different components of SwitchML or of the system as a whole.

The benchmarks should be the go-to tool that ensures that the performance and the accuracy of SwitchML remains as expected after any change.

## 7.1 1. Benchmarks list

| Benchmark | Brief |
|---|---|
| allreduce_benchmark | The most complete benchmark, as it actually performs allreduce jobs thus testing the whole system. |

All examples require that the client library is compiled and that the SwitchML configuration file is present when running.

## 7.2 2. Compiling

To compile a benchmark, simply run (Assuming you are inside the benchmarks directory):

```
make <benchmark_name> [build variables]
```

Or to compile all benchmarks at once just run

```
make [build variables]
```

By default, the benchmark executables will be found in

```
dev_root/build/bin/
```

### 7.2.1 2.1 Build variables

The following variables can all be passed to the benchmarks makefile to control the build.

| Variable | Type | Default | Usage |
|---|---|---|---|
| DEBUG | boolean | 0 | Disable optimizations, add debug symbols. |
| DPDK | boolean | 0 | Add dpdk backend specific compiler/linker options. |
| MLX5 | boolean | 0 | Add dpdk backend Connect-x5/Connect-x4 specific compiler/linker options. |
| MLX4 | boolean | 0 | Add dpdk backend Connect-x3 specific compiler/linker options. |
| RDMA | boolean | 0 | Add rdma backend specific compiler/linker options. |
| CUDA | boolean | 0 | Compile benchmark with cuda support. Allows allocating tensors on the gpu and passing their pointers to the client library. (Do not use as GPU memory is not yet handled by any of the prepostprocessors in the client library) |
| BUILD-DIR | path | dev_root/build | Where to store generated objects/include files/libraries/binaries... etc. |
| SWITCHML_HOME | path | dev_root/build | Where to look for the switchml client library installation |
| GRPC_HOME | path | dev_root/third_party/grpc/build | Where to look for the GRPC installation |
| DPDK_HOME | path | dev_root/third_party/dpdk | Where to look for the DPDK installation |

CHAPTER

EIGHT

# FRAMEWORKS INTEGRATION

In order to use **SwitchML** with one of the popular DNN frameworks such as Tensorflow or PyTorch, you need to use one of the integration methods.

| Integration method | Supported Frameworks | Status |
|---|---|---|
| NCCL Plugin | Tensorflow through Horovod, PyTorch | Needs more testing |
| Pytorch Patch | Pytorch | Stable |

Read more about each method by checking its corresponding documentation.

CHAPTER

NINE

# SCRIPTS

The scripts directory contains various bash and python helper scripts.

| Script | Description |
|---|---|
| disable-icrc.sh | disables ICRC checking for a specific ibverbs device. This is needed for the client with rdma backend to work properly. Make sure to examine the script and customize it to your device name and your setup in general |
| enable-icrc.sh | enables ICRC checking back. |

CHAPTER

TEN

# CONTRIBUTING

The SwitchML project welcomes all contributions. But before trying to contribute please read through the following guidelines specifically the ones concerning the part that you would like to tackle. It will make you understand the design, style, and conventions used faster and get you up to speed on what you should adhere to and what you should be aware of when contributing.

After doing that, please open up an issue on Github whether you wanted to fix a bug, implement a new feature... etc describing exactly what you plan to do. This will allow the team to give you feedback and discuss the various dimensions of the contribution. It will also make it all the more likely for your contribution to be pulled into the main repo.

**Table of contents**


- *General Guidelines*
  - *Coding Style Conventions*
- *Client Library*
  - *How the different classes interact*
  - *Logging*
- *P4*
- *Controller*
- *Examples*
- *Benchmarks*
- *Documentation*


## 10.1 General Guidelines

When proposing a code change or adding a new feature, one must take into account all of the code that may have relied on the old code and update it accordingly.

This is why API and interface changes of the different components/classes can be much more difficult to add and requires lots of vigilance. Changing something in the P4 program may require changes in both the client library backend and the python controller. Changing something in the client library context may require changes to all framework integration methods and many examples and benchmarks. So it is important to keep all of these dependencies in mind when changing how a component interfaces with other components.

On the other hand, editing the implementation of components / functions to either fix a bug, improve performance, or enhance readability is much easier to accommodate and accept.

Similarly, adding new features or functions that do not break any of the other components is also easy to accommodate and include given enough justification for the feature and showing that it is within the scope of SwitchML.

### 10.1.1 Coding Style Conventions

When writing code for SwitchML, there is a few conventions that you should adhere to to ensure consistency and readability across the project depending on the type of file that you are working on.

### 10.1.2 Header

We adopt the following header for all types of files

```
# SwitchML Project
# @file the name of the file
# @brief a one line description of the file
#
# More details if needed
```

Whether you use `#`, or doxygen's `/** */` depends on the type of file that you are documenting.

#### C++

For documentation we adopt the javadoc doxygen style using `/** */` to enable us to generate automatic documentation. For everything else we adopt the Google C++ Style with the exception of indentation where we use 4 spaces for each level as we feel that improves readability.

#### Python

...

#### P4

...

## 10.2 SwitchML Client Library

### 10.2.1 How the different classes interact

Everything starts with the **Context** class. The Context is a singleton class that provides all SwitchML services to the user through its simplified API. Firstly, a user gets an instance of the context then starts it through the *Start()* function. During initialization the following occurs:

1. A **Config** object is created to represent all of the options specified in the `switchml.cfg` file. Optionally you can create this object yourself and pass it to the **Context** to avoid using configuration files.
2. A **Backend** object is created (dependending on the backend chosen in the **Config**) which typically starts **WorkerThread**s that poll the context for work and carry it out.
3. A **Scheduler** object is created (dependending on the scheduler chosen in the **Config**) which the context uses to dispatch **JobSlice**s to the **WorkerThread**s.

After initialization we will have a number of Backend specific **WorkerThread**s (depending on the number set in the **Config**) constantly asking the **Context** for work. A user can then submit work (Ex. an all reduce operation) through the **Context**. The **Context** then creates a **Job** out of this function call and submits it to its internal **Scheduler**. When the continuously running **WorkerThread**s ask the **Context** for work to do, the **Job** is divided into **JobSlice**s by the scheduler which are then dispatched to the **WorkerThread**s. Each **WorkerThread** receives a **JobSlice** then it performs the work necessary to perform the actual communication. Finally the **WorkerThread** notifies the **Context** when the **JobSlice** work is completed. When all **JobSlice**s of a **Job** are completed, the context notifies any threads waiting on this **Job**.

**WorkerThread**s utilize a **PrePostProcessor** to load and unload the data from and into packets or messages. The **WorkerThread** never actually reads the client's data or writes into the client's buffers. That is the sole job of the **PrePostProcessor**

### 10.2.2 Logging

For logging we use the glog library

We have specific conventions for logging so make sure that you adhere to these conventions when writing your code. If you have a convincing reason on why you did not, please clarify that in your pull request.

| Severity | Usage |
|---|---|
| INFO | General information that the user might be interested in. Refer to the verbosity table for more details as we never use `LOG(INFO)` but instead `VLOG(verbosity)` for logging informational messages |
| WARNING | Alert the user to a potential problem in his configuration or setup |
| ERROR | Notify the user about a definite problem in their configuration or environment but SwitchML can remedy the error and continue on |
| FATAL | Notify the user about a serious unrecoverable problem that occurred after which the application will exit. We use both `LOG(FATAL)` and `CHECK(condition)` to signal problems of this magnitude |

We include all logging statements with severity higher than INFO in both the debugging and release builds. For the informational messages, whether they are included or not depends on the verbosity as will be illustrated in what follows.

| Verbosity | Usage |
|---|---|
| 0 | General initialization and cleanup information mostly produced by the context itself that all SwitchML users can understand |
| 1 | Initialization and cleanup information of all classes in the client library. This includes backend initialization and cleanup, worker threads initialization and cleanup, dumping usage stats..etc. |
| 2 | Information about job submission, scheduling, and job completion. |
| 3 | Information about packets and messages being sent and received |
| 4 | Information about the contents of the data itself that is being processed |

For verbosity levels <= 1, you should use `VLOG` since initialization and cleanup will not affect performance during the lifetime of the application. On the other hand, for verbosity levels > 1, you should use `DVLOG` since we do not want these statements to be included except in a DEBUG build as they can affect performance significantly.

## 10.3 SwitchML P4 Program

...

## 10.4 SwitchML Controller

...

## 10.5 Examples

Examples are meant to be step by step documented small programs to illustrate the usage of some component of switchml. We always welcome new well written and well documented examples that make understanding the components of the library a little more easier. We also welcome improvements of existing examples whether in documentation or the code/comments themselves.

## 10.6 Benchmarks

Benchmarks are meant to measure the speed and correctness of switchml as a whole or for specific parts of it (For ex. schedulers, prepostprocessors...etc.). We always welcome new benchmarks that help us evaluate the different parts of switchml in terms of speed and correctness. We also welcome improvements of existing examples whether in documentation or the code/comments themselves.

## 10.7 Documentation

Contributing to the documentation is always welcome. If you found an error, feel that you can explain something in a better way, or wanted to add new guides/pages that will help new users/developers understand switchml, then by all means don't hesitate to write your changes and open a pull request. However we ask that you try to adhere to the implicit conventions and 'feel' of the existing documentation.

Switchml is still in its early development stages and documentation can become outdated quite fast. While us the developers can get accustomed and miss some of these outdated portions, new users will certainly not, which makes a new user (perhaps yourself) a very important asset in helping us keep the documentation up to date.

# INDEX


## D

DPDK_SWITCH_ELEMENT_SIZE (*C macro*), 63
DUMMY (*C macro*), 63
DUMMY_ELEMENT_SIZE (*C macro*), 63

## R

RDMA_SWITCH_ELEMENT_SIZE (*C macro*), 63

## S

switchml::AllReduceOperation (*C++ enum*), 55
switchml::AllReduceOperation::SUM (*C++ enumerator*), 55
switchml::Backend (*C++ class*), 22
switchml::Backend::~Backend (*C++ function*), 22
switchml::Backend::Backend (*C++ function*), 22
switchml::Backend::CleanupWorker (*C++ function*), 22
switchml::Backend::config_ (*C++ member*), 23
switchml::Backend::context_ (*C++ member*), 23
switchml::Backend::CreateInstance (*C++ function*), 22
switchml::Backend::operator= (*C++ function*), 22
switchml::Backend::SetupWorker (*C++ function*), 22
switchml::BackendConfig (*C++ struct*), 14
switchml::BackendConfig::dpdk (*C++ member*), 14
switchml::BackendConfig::rdma (*C++ member*), 14
switchml::Barrier (*C++ class*), 23
switchml::Barrier::~Barrier (*C++ function*), 23
switchml::Barrier::Barrier (*C++ function*), 23
switchml::Barrier::Destroy (*C++ function*), 23
switchml::Barrier::operator= (*C++ function*), 23
switchml::Barrier::Wait (*C++ function*), 23
switchml::BindToCore (*C++ function*), 57
switchml::BypassPPP (*C++ class*), 24
switchml::BypassPPP::~BypassPPP (*C++ function*), 24
switchml::BypassPPP::BypassPPP (*C++ function*), 24
switchml::BypassPPP::CleanupJobSlice (*C++ function*), 25
switchml::BypassPPP::NeedsExtraBatch (*C++ function*), 24
switchml::BypassPPP::operator= (*C++ function*), 24
switchml::BypassPPP::PostprocessSingle (*C++ function*), 25
switchml::BypassPPP::PreprocessSingle (*C++ function*), 24
switchml::BypassPPP::SetupJobSlice (*C++ function*), 24
switchml::ChangeMacEndianness (*C++ function*), 57
switchml::clock (*C++ type*), 64
switchml::Config (*C++ class*), 25
switchml::Config::~Config (*C++ function*), 25
switchml::Config::backend_ (*C++ member*), 26
switchml::Config::Config (*C++ function*), 25
switchml::Config::general_ (*C++ member*), 26
switchml::Config::LoadFromFile (*C++ function*), 25
switchml::Config::operator= (*C++ function*), 25
switchml::Config::PrintConfig (*C++ function*), 25
switchml::Config::Validate (*C++ function*), 25
switchml::Context (*C++ class*), 26
switchml::Context::AllReduce (*C++ function*), 28

switchml::Context::AllReduceAsync (C++ function), 27
switchml::Context::Context (C++ function), 27
switchml::Context::ContextState (C++ enum), 26
switchml::Context::ContextState::CREATED (C++ enumerator), 26
switchml::Context::ContextState::RUNNING (C++ enumerator), 26
switchml::Context::ContextState::STARTING (C++ enumerator), 26
switchml::Context::ContextState::STOPPED (C++ enumerator), 26
switchml::Context::ContextState::STOPPING (C++ enumerator), 26
switchml::Context::GetConfig (C++ function), 28
switchml::Context::GetContextState (C++ function), 28
switchml::Context::GetInstance (C++ function), 28
switchml::Context::GetStats (C++ function), 28
switchml::Context::operator= (C++ function), 27
switchml::Context::Start (C++ function), 27
switchml::Context::Stop (C++ function), 27
switchml::Context::WaitForAllJobs (C++ function), 28
switchml::CpuExponentQuantizerPPP (C++ class), 29
switchml::CpuExponentQuantizerPPP::~CpuExponentQuantizerPPP (C++ function), 29
switchml::CpuExponentQuantizerPPP::CleanupJobSlice (C++ function), 30
switchml::CpuExponentQuantizerPPP::CpuExponentQuantizerPPP (C++ function), 29
switchml::CpuExponentQuantizerPPP::NeedsExtraBandwidth (C++ function), 29
switchml::CpuExponentQuantizerPPP::operator= (C++ function), 29
switchml::CpuExponentQuantizerPPP::PostprocessSingle (C++ function), 30
switchml::CpuExponentQuantizerPPP::PreprocessSingle (C++ function), 30
switchml::CpuExponentQuantizerPPP::SetupJobSlice (C++ function), 29
switchml::DataType (C++ enum), 56
switchml::DataType::FLOAT32 (C++ enumerator), 56
switchml::DataType::INT32 (C++ enumerator), 56
switchml::DataTypeSize (C++ function), 58
switchml::DpdkBackend (C++ class), 31
switchml::DpdkBackend::~DpdkBackend (C++ function), 31
switchml::DpdkBackend::CleanupWorker (C++ function), 31
switchml::DpdkBackend::DpdkBackend (C++ function), 31
switchml::DpdkBackend::DpdkPacketElement (C++ type), 31
switchml::DpdkBackend::DpdkPacketHdr (C++ struct), 14, 32
switchml::DpdkBackend::DpdkPacketHdr::job_type_size (C++ member), 14, 32, 62
switchml::DpdkBackend::DpdkPacketHdr::pkt_id (C++ member), 14, 32, 62
switchml::DpdkBackend::DpdkPacketHdr::short_job_id (C++ member), 14, 32, 62
switchml::DpdkBackend::DpdkPacketHdr::switch_pool_i (C++ member), 14, 32, 62
switchml::DpdkBackend::E2eAddress (C++ struct), 15, 32
switchml::DpdkBackend::E2eAddress::ip (C++ member), 15, 33
switchml::DpdkBackend::E2eAddress::mac (C++ member), 15, 33
switchml::DpdkBackend::E2eAddress::port (C++ member), 15, 33
switchml::DpdkBackend::GetSwitchE2eAddr (C++ function), 32
switchml::DpdkBackend::GetWorkerE2eAddr (C++ function), 32
switchml::DpdkBackend::GetWorkerThreads (C++ function), 32
switchml::DpdkBackend::operator= (C++ function), 31
switchml::DpdkBackend::SetupSwitch (C++ function), 32
switchml::DpdkBackend::SetupWorker (C++ function), 31
switchml::DpdkBackendConfig (C++ struct), 15
switchml::DpdkBackendConfig::bulk_drain_tx_us (C++ member), 16
switchml::DpdkBackendConfig::burst_rx (C++ member), 16
switchml::DpdkBackendConfig::burst_tx (C++ member), 16
switchml::DpdkBackendConfig::cores_str (C++ member), 15
switchml::DpdkBackendConfig::extra_eal_options (C++ member), 16
switchml::DpdkBackendConfig::pool_cache_size (C++ member), 16
switchml::DpdkBackendConfig::pool_size

(C++ member), 16
switchml::DpdkBackendConfig::port_id (C++ member), 16
switchml::DpdkBackendConfig::worker_ip_str (C++ member), 15
switchml::DpdkBackendConfig::worker_port (C++ member), 15
switchml::DpdkMasterThread (C++ class), 33
switchml::DpdkMasterThread::~DpdkMasterThread (C++ function), 33
switchml::DpdkMasterThread::DpdkMasterThread (C++ function), 33
switchml::DpdkMasterThread::Join (C++ function), 33
switchml::DpdkMasterThread::operator() (C++ function), 33
switchml::DpdkMasterThread::operator= (C++ function), 33
switchml::DpdkMasterThread::Start (C++ function), 33
switchml::DpdkWorkerThread (C++ class), 34
switchml::DpdkWorkerThread::~DpdkWorkerThread (C++ function), 34
switchml::DpdkWorkerThread::DpdkWorkerThread (C++ function), 34
switchml::DpdkWorkerThread::operator() (C++ function), 34
switchml::DpdkWorkerThread::operator= (C++ function), 34
switchml::DpdkWorkerThread::tid_ (C++ member), 34
switchml::DummyBackend (C++ class), 35
switchml::DummyBackend::~DummyBackend (C++ function), 35
switchml::DummyBackend::CleanupWorker (C++ function), 36
switchml::DummyBackend::CleanupWorkerThread (C++ function), 36
switchml::DummyBackend::DummyBackend (C++ function), 35
switchml::DummyBackend::DummyPacket (C++ struct), 17, 36
switchml::DummyBackend::DummyPacket::data_type (C++ member), 17, 37
switchml::DummyBackend::DummyPacket::entries_num (C++ member), 17, 37
switchml::DummyBackend::DummyPacket::ext_info_ptr (C++ member), 17, 37
switchml::DummyBackend::DummyPacket::job_id (C++ member), 17, 37
switchml::DummyBackend::DummyPacket::numel (C++ member), 17, 37
switchml::DummyBackend::DummyPacket::pkt_id (C++ member), 17, 37
switchml::DummyBackend::operator= (C++ function), 35
switchml::DummyBackend::ReceiveBurst (C++ function), 36
switchml::DummyBackend::SendBurst (C++ function), 36
switchml::DummyBackend::SetupWorker (C++ function), 35
switchml::DummyBackend::SetupWorkerThread (C++ function), 36
switchml::DummyWorkerThread (C++ class), 37
switchml::DummyWorkerThread::~DummyWorkerThread (C++ function), 37
switchml::DummyWorkerThread::DummyWorkerThread (C++ function), 37
switchml::DummyWorkerThread::Join (C++ function), 38
switchml::DummyWorkerThread::operator() (C++ function), 37
switchml::DummyWorkerThread::operator= (C++ function), 37
switchml::DummyWorkerThread::Start (C++ function), 37
switchml::DummyWorkerThread::tid_ (C++ member), 38
switchml::Execute (C++ function), 58
switchml::ExtraJobInfo (C++ union), 57
switchml::ExtraJobInfo::allreduce_operation (C++ member), 57
switchml::ExtraJobInfo::broadcast_root_rank (C++ member), 57
switchml::FifoScheduler (C++ class), 38
switchml::FifoScheduler::~FifoScheduler (C++ function), 38
switchml::FifoScheduler::EnqueueJob (C++ function), 38
switchml::FifoScheduler::FifoScheduler (C++ function), 38
switchml::FifoScheduler::GetJobSlice (C++ function), 39
switchml::FifoScheduler::NotifyJobSliceCompletion (C++ function), 39
switchml::FifoScheduler::operator= (C++ function), 38
switchml::FifoScheduler::Stop (C++ function), 39
switchml::GeneralConfig (C++ struct), 17
switchml::GeneralConfig::backend (C++ member), 18
switchml::GeneralConfig::controller_ip_str (C++ member), 18
switchml::GeneralConfig::controller_port (C++ member), 18
switchml::GeneralConfig::instant_job_completion

(C++ member), 18
switchml::GeneralConfig::max_outstanding_packets (C++ member), 17
switchml::GeneralConfig::num_worker_threads (C++ member), 17
switchml::GeneralConfig::num_workers (C++ member), 17
switchml::GeneralConfig::packet_numel (C++ member), 18
switchml::GeneralConfig::prepostprocessor (C++ member), 18
switchml::GeneralConfig::rank (C++ member), 17
switchml::GeneralConfig::scheduler (C++ member), 18
switchml::GeneralConfig::timeout (C++ member), 18
switchml::GeneralConfig::timeout_threshold (C++ member), 18
switchml::GeneralConfig::timeout_threshold_increment (C++ member), 18
switchml::GetCoresNuma (C++ function), 58
switchml::GetDeviceNuma (C++ function), 59
switchml::GIDToIPv4 (C++ function), 59
switchml::GIDToMAC (C++ function), 59
switchml::GrpcClient (C++ class), 40
switchml::GrpcClient::~GrpcClient (C++ function), 40
switchml::GrpcClient::Barrier (C++ function), 40
switchml::GrpcClient::Broadcast (C++ function), 40
switchml::GrpcClient::CreateRdmaSession (C++ function), 40
switchml::GrpcClient::CreateUdpSession (C++ function), 40
switchml::GrpcClient::GrpcClient (C++ function), 40
switchml::GrpcClient::operator= (C++ function), 40
switchml::IPv4ToGID (C++ function), 60
switchml::Job (C++ class), 41
switchml::Job::~Job (C++ function), 41
switchml::Job::extra_job_info_ (C++ member), 42
switchml::Job::GetJobStatus (C++ function), 41
switchml::Job::id_ (C++ member), 42
switchml::Job::Job (C++ function), 41
switchml::Job::job_type_ (C++ member), 42
switchml::Job::operator= (C++ function), 41
switchml::Job::SetJobStatus (C++ function), 41
switchml::Job::tensor_ (C++ member), 42
switchml::Job::WaitToComplete (C++ function), 41
switchml::JobId (C++ type), 64
switchml::JobSlice (C++ struct), 18
switchml::JobSlice::job (C++ member), 19
switchml::JobSlice::slice (C++ member), 19
switchml::JobStatus (C++ enum), 56
switchml::JobStatus::FAILED (C++ enumerator), 56
switchml::JobStatus::FINISHED (C++ enumerator), 56
switchml::JobStatus::INIT (C++ enumerator), 56
switchml::JobStatus::QUEUED (C++ enumerator), 56
switchml::JobStatus::RUNNING (C++ enumerator), 56
switchml::JobType (C++ enum), 56
switchml::JobType::ALLREDUCE (C++ enumerator), 56
switchml::JobType::BROADCAST (C++ enumerator), 56
switchml::Mac2Str (C++ function), 60
switchml::MACToGID (C++ function), 61
switchml::Numel (C++ type), 64
switchml::PrePostProcessor (C++ class), 42
switchml::PrePostProcessor::~PrePostProcessor (C++ function), 42
switchml::PrePostProcessor::batch_max_num_ltus_ (C++ member), 44
switchml::PrePostProcessor::CleanupJobSlice (C++ function), 43
switchml::PrePostProcessor::config_ (C++ member), 44
switchml::PrePostProcessor::CreateInstance (C++ function), 44
switchml::PrePostProcessor::ltu_size_ (C++ member), 44
switchml::PrePostProcessor::NeedsExtraBatch (C++ function), 43
switchml::PrePostProcessor::operator= (C++ function), 42
switchml::PrePostProcessor::PostprocessSingle (C++ function), 43
switchml::PrePostProcessor::PrePostProcessor (C++ function), 42, 44
switchml::PrePostProcessor::PreprocessSingle (C++ function), 43
switchml::PrePostProcessor::SetupJobSlice (C++ function), 42
switchml::PrePostProcessor::worker_tid_ (C++ member), 44
switchml::RdmaBackend (C++ class), 45
switchml::RdmaBackend::~RdmaBackend

(C++ function), 45
switchml::RdmaBackend::CleanupWorker (C++ function), 45
switchml::RdmaBackend::GetConnection (C++ function), 45
switchml::RdmaBackend::operator= (C++ function), 45
switchml::RdmaBackend::RdmaBackend (C++ function), 45
switchml::RdmaBackend::SetupWorker (C++ function), 45
switchml::RdmaBackendConfig (C++ struct), 19
switchml::RdmaBackendConfig::device_names (C++ member), 19
switchml::RdmaBackendConfig::device_ports (C++ member), 19
switchml::RdmaBackendConfig::gid_index (C++ member), 19
switchml::RdmaBackendConfig::msg_numel (C++ member), 19
switchml::RdmaBackendConfig::use_gdr (C++ member), 19
switchml::RdmaConnection (C++ class), 46
switchml::RdmaConnection::~RdmaConnection (C++ function), 46
switchml::RdmaConnection::Connect (C++ function), 46
switchml::RdmaConnection::GetEndpoint (C++ function), 47
switchml::RdmaConnection::GetWorkerThreadCompletionQueue (C++ function), 46
switchml::RdmaConnection::GetWorkerThreadMemoryRegion (C++ function), 46
switchml::RdmaConnection::GetWorkerThreadQueuePairs (C++ function), 46
switchml::RdmaConnection::GetWorkerThreadRkeys (C++ function), 46
switchml::RdmaConnection::operator= (C++ function), 46
switchml::RdmaConnection::PostRecv (C++ function), 47
switchml::RdmaConnection::PostSend (C++ function), 47
switchml::RdmaConnection::RdmaConnection (C++ function), 46
switchml::RdmaEndpoint (C++ class), 47
switchml::RdmaEndpoint::~RdmaEndpoint (C++ function), 47
switchml::RdmaEndpoint::AllocateAtAddress (C++ function), 47
switchml::RdmaEndpoint::CreateCompletionQueue (C++ function), 48
switchml::RdmaEndpoint::CreateQueuePair (C++ function), 48
switchml::RdmaEndpoint::free (C++ function), 48
switchml::RdmaEndpoint::GetDevice (C++ function), 48
switchml::RdmaEndpoint::GetIPv4 (C++ function), 48
switchml::RdmaEndpoint::GetMac (C++ function), 48
switchml::RdmaEndpoint::GetPortAttributes (C++ function), 48
switchml::RdmaEndpoint::operator= (C++ function), 47
switchml::RdmaEndpoint::RdmaEndpoint (C++ function), 47
switchml::RdmaWorkerThread (C++ class), 48
switchml::RdmaWorkerThread::~RdmaWorkerThread (C++ function), 49
switchml::RdmaWorkerThread::Join (C++ function), 49
switchml::RdmaWorkerThread::operator() (C++ function), 49
switchml::RdmaWorkerThread::operator= (C++ function), 49
switchml::RdmaWorkerThread::RdmaWorkerThread (C++ function), 49
switchml::RdmaWorkerThread::Start (C++ function), 49
switchml::RdmaWorkerThread::tid_ (C++ member), 49
switchml::Scheduler (C++ class), 50
switchml::Scheduler::~Scheduler (C++ function), 50
switchml::Scheduler::access_mutex_ (C++ member), 51
switchml::Scheduler::config_ (C++ member), 51
switchml::Scheduler::CreateInstance (C++ function), 51
switchml::Scheduler::EnqueueJob (C++ function), 50
switchml::Scheduler::GetJobSlice (C++ function), 50
switchml::Scheduler::job_submitted_event_ (C++ member), 51
switchml::Scheduler::NotifyJobSliceCompletion (C++ function), 50
switchml::Scheduler::operator= (C++ function), 50
switchml::Scheduler::Scheduler (C++ function), 50, 51
switchml::Scheduler::Stop (C++ function), 51
switchml::Scheduler::stopped_ (C++ member), 51

switchml::Stats (C++ class), 52
switchml::Stats::~Stats (C++ function), 52
switchml::Stats::AddCorrectPktsReceived (C++ function), 53
switchml::Stats::AddTimeouts (C++ function), 53
switchml::Stats::AddTotalPktsSent (C++ function), 53
switchml::Stats::AddWrongPktsReceived (C++ function), 53
switchml::Stats::AppendJobSubmittedNumel (C++ function), 53
switchml::Stats::DescribeFloatList (C++ function), 52
switchml::Stats::DescribeIntList (C++ function), 52
switchml::Stats::IncJobsFinishedNum (C++ function), 53
switchml::Stats::IncJobsSubmittedNum (C++ function), 53
switchml::Stats::InitStats (C++ function), 52
switchml::Stats::List2Str (C++ function), 52
switchml::Stats::LogStats (C++ function), 52
switchml::Stats::operator= (C++ function), 52
switchml::Stats::ResetStats (C++ function), 52
switchml::Stats::Stats (C++ function), 52
switchml::Str2Mac (C++ function), 61
switchml::Tensor (C++ struct), 20
switchml::Tensor::data_type (C++ member), 20
switchml::Tensor::in_ptr (C++ member), 20
switchml::Tensor::numel (C++ member), 20
switchml::Tensor::OffsetPtrs (C++ function), 20
switchml::Tensor::out_ptr (C++ member), 20
switchml::TimeoutQueue (C++ class), 53
switchml::TimeoutQueue::~TimeoutQueue (C++ function), 54
switchml::TimeoutQueue::Check (C++ function), 54
switchml::TimeoutQueue::operator= (C++ function), 54
switchml::TimeoutQueue::Push (C++ function), 54
switchml::TimeoutQueue::Remove (C++ function), 54
switchml::TimeoutQueue::TimeoutQueue (C++ function), 54
switchml::TimeoutQueue::TimePoint (C++ type), 53
switchml::TimeoutQueue::TQEntry (C++ struct), 21, 54
switchml::TimeoutQueue::TQEntry::~TQEntry (C++ function), 21, 55
switchml::TimeoutQueue::TQEntry::next (C++ member), 21, 55
switchml::TimeoutQueue::TQEntry::operator= (C++ function), 21, 55
switchml::TimeoutQueue::TQEntry::previous (C++ member), 21, 55
switchml::TimeoutQueue::TQEntry::timestamp (C++ member), 21, 55
switchml::TimeoutQueue::TQEntry::TQEntry (C++ function), 21, 55
switchml::TimeoutQueue::TQEntry::valid (C++ member), 21, 55
switchml::ToHex (C++ function), 61
switchml::WorkerTid (C++ type), 64