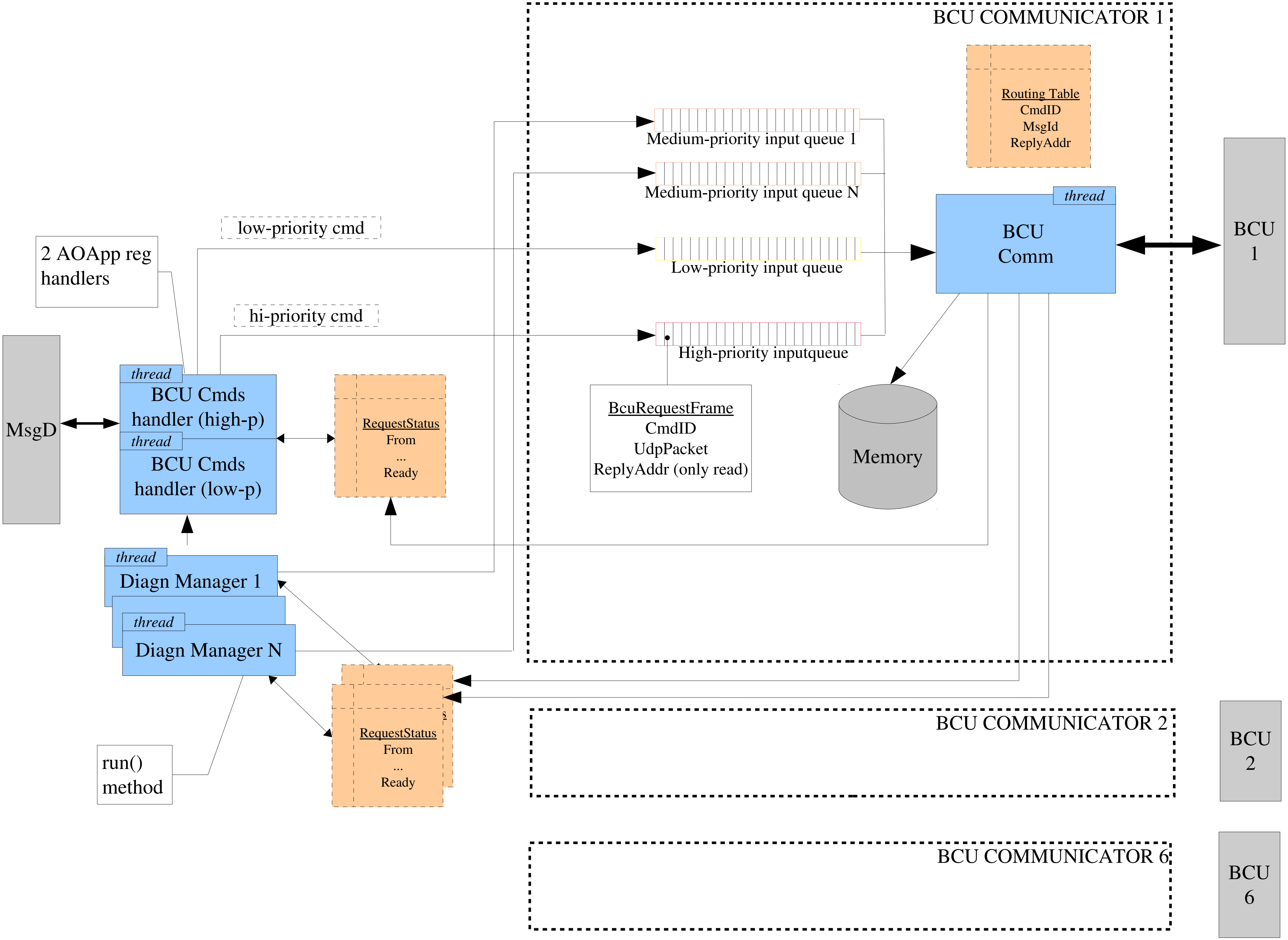
Mirror Controller 2.0 – Architecture



Main modules [not updated respect to diagram]

References: *MirrorCtrl-Requirements.odt* for functional requirements (fr-x) and not-functional requirements (nfr-y).

MirrorCtrl extends AOApp (nfr-b)

BCU Cmds Handler (2 AOApp notify thread)

- ✔ A command request is blocking (the thread will wait for the reply), to simplify the thread and to respect requirements (fr-a.1.3.2).
- ✔ Low-priority and high-priority commands are managed by two different thread, because a command request is blocking and don't want to block an high-priority command while a low-priority one is performing (fr-a.1.3.1).
- ✔ The two threads are identical: they only fill different input-queues !!! The high-priority commands thread could have a bigger priority.
- ✔ It manages MsgD messages (BCU commands) and UDP packets (fr-a.2).
- ✔ It takes care about messages routing (both directions) (fr-a.2, fr-a.3).

(A) It receives BCU commands from MsgD and forward them to the correct BCUs (or broadcast) (fr-a.2).

1. Receive message (MsgBuf) from MsgD (automatically performed by thrdlib)
2. Generate a unique CmdID
3. Generate UDP packets and put them in the corresponding priority queues (hi/low priority)
4. Update a structure to bind CmdID with message sender (the sender will always be the receiver of the reply!)
5. Blocks itself, waiting for the reply.

(B) Get replies from BCU (via BCU Comm and queue) and forward them to MsgD (fr-a.3).

1. Pop UDP packets from the output queue
2. Unpack them and build the message
3. Forward to the correct destination via MsgD (using the binding created at step A.3)

Notes:

- ✗ Also if commands are not-blocking, can't be a single thread: steps A are automatically performed when the thrdlib notify it. When steps B are performed ?

  Possible solution

```
class BcuCmdHandler {
public:
      BcuCmdHandler();
      static RequestHandler(); //Register this for MsgD notify, performed by thrdlib
      void StartReplyHandler() {
            pthread_create(&_replyThread, NULL, ReplyHandler, this);
      }

private:
      pthread_t _replyThread;
```

```
    void ReplyHandler(void *pthis) { //Loop polling reply queue }
}
```

MirrorCtrl instantiate a BcuCmdHandler and:

- Register `BcuCmdHandler::RequestHandler` for messages from the MsgD
- Start the `BcuCmdHandler` thread for managing replies from BCU

-> This solution is more complicated, because the receiving thread must pop two different queues !!! This solution isn't also fair: it will block an high-priority command while a low-priority command is sending !!!

Implementation possibilities

- The 2 threads are automatically implemented by thrdlib using the register/notify feature.
- Each thread main module is a class providing (aggregation relation) an input queue and an output queue.
- The thread priority (corresponding to queue priority and, if needed, to the thread priority), is defined in the class constructor.

Diagnostic Manager

There is a set of N Diagnostic Manager threads, one for each *diagnostic task* defined in the configuration file (usually 2, see fr-b.4.1). A Diagnostic Manager is coupled with a **BCU Comm** thread by mean of two queues: one (medium-priority) input-queue and one output-queue (direction is meant respect to BCU Comm) (nfr-c, nfr-d).
Each thread perform diagnostic frames downloading from the pool of N BCUs, that is, it communicates with N BCU Comm (fr-b.1, fr-b.2). In total, a Diagnostic Manager thread uses N input-queues and N output-queues.

1 Gets the current frame counter from BCU 1
   1.1 Creates the request packet (single UDP packet)
   1.2 Puts it in the input-queue-1
   1.3 Gets the reply packet

2 Generates diagnostic requests for each BCU
   - Creates K UDP packets
   - Put them into all N input-queues (one for each BCUs)

3 Receives N diagnostic frames (UDP packets): for each of N output queues
   - Waits for packets replies
   - Unpack them generating the partial frame
   - Copy the partial frame in a local buffer

4 Reconstructs the diagnostic full frame (automatically done in previous step)

5 Performs a bufWrite to shared-memory.

Implementation possibilities

- The queues are created (aggregation) by *Diagnostic Manager* and not by *BCU Comm* because the first one know how many BCUs exists; *BCU Comm* thread doesn't have any reason to know how many “brothers” he have.
- A Diagnostic Manager thread is spawned in the AOApp run method:
  - ➔ A Diagnostic Manager object (*diagnMan*) is created: it creates its own input-queue and output-queue pools.
  - ➔ Two Diagnostic Manager object methods allow to get the references to the 2 pools of queues, to pass a pair of in/out queues to the BCU Comm thread (i.e. GetInputQPool(), get OutputQPool())
  - ➔ The thread is started using the static method StartDiagnostic(DiagnosticManager *diagnMan*) of the MirrorCtrl.
- ...

BCU Comm (thread)

- ✔ It manages only UDP packets
- ✔ It doesn't care about UDP packet's content
- ✔ It stores a counter with the BCU packets availability

A. Forward UDP packets from the 2+K (K is the number of input-queues, that is, the number of Diagnostic Managers) priority input-queues to BCU, using an appropriate scheduler (i.e. Round-robin, but a scheduler taking care of queues priorities should be better).

1. Get an UDP packet from a queue
2. Save the binding CmdID->Output-Queue for the packet (if doesn't already exists), considering the on input-queue. I.e. If the packet with CmdID=X has been got from input-queue-j, all reply packets with CmdID=X must be put in output-queue-j.
3. Send the UDP packet to BCU

B. Receive UDP packets from BCU (replies). Forward diagnostic frames packets (UDP) to *Diagn Manager*; forward others replies to MsgD.

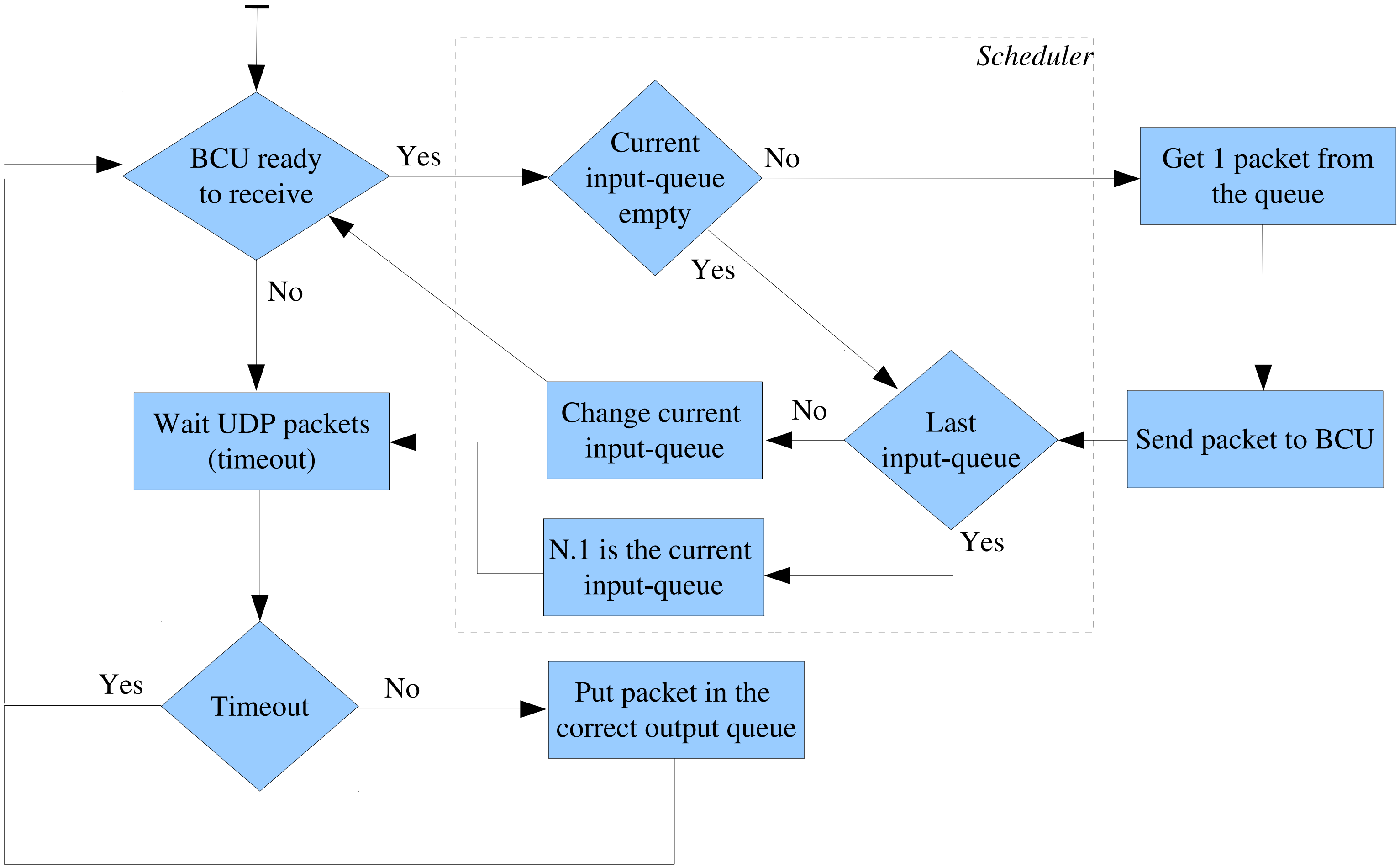
1. Receive a reply UDP packet from BCU
2. Check the binding CmdID->Output-Queue (see A.2) and forward to the correct queue.

<u>Guidelines</u>

- ✔ To speed-up the BCU throughput is needing:
  - ● The input-queue must be always full to obtain the maximum throughput
  - ● The output queue must not overflow (should be rather empty)
    -> Because the Ethernet is slow, and the threads are fast, the output queue should be never overflow. For the same reason the Diagnostic Managers shouldn'0t have any problem to fulfill the input queues !!!

<u>Implementation possibilities</u>

- ◆ The BCU Comm thread is spawned in the AOApp run method, or is the AOApp main thread.
- ◆ BCU Comm have a method to add (attach) an input/output pair of queues (also at runtime ?!)


Scheduler
BCU ready to receive
Yes
No
Current input-queue empty
No
Yes
Get 1 packet from the queue
Send packet to BCU
Last input-queue
No
Yes
Change current input-queue
N.1 is the current input-queue
Wait UDP packets (timeout)
Timeout
Yes
No
Put packet in the correct output queue