Anna Belova, JINR, 2020.

Prospects to use the FairMQ data exchange system for SPD

SPD ROOT

Monte Carlo simulation, event reconstruction for both simulated and real data, data analysis and visualization are planned to be performed by an object oriented C++ toolkit SPDroot. It is based on the FairRoot framework initially developed for the FAIR experiments at GSI Darmstadt and partially compatible with MPDroot and BM@Nroot software used at MPD and BM@N, respectively.

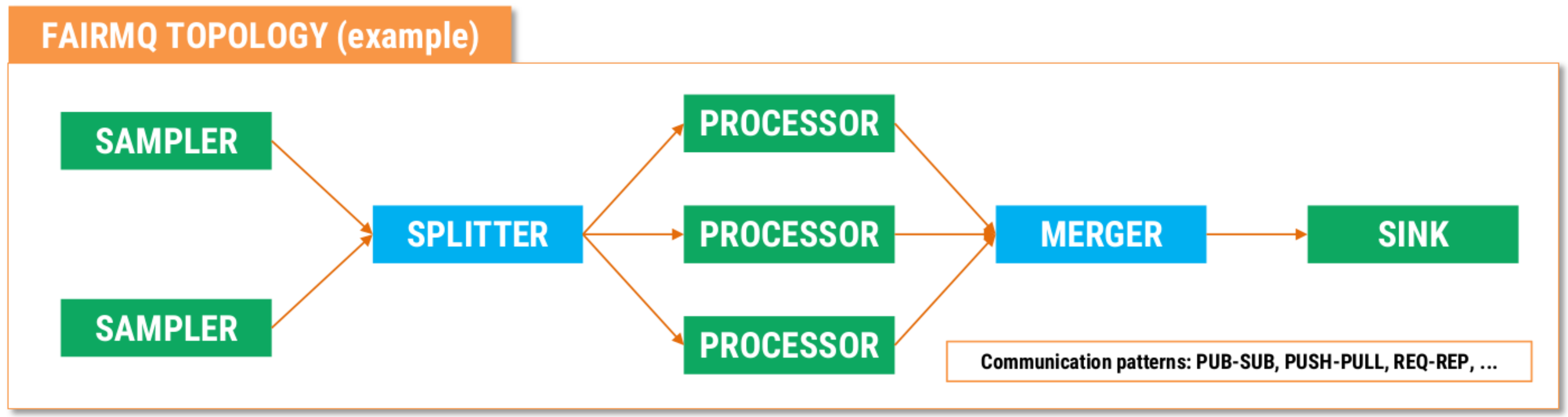
The SPD detector description for Monte Carlo simulation is based on the ROOT geometry while transportation of secondary particles through material of the setup and simulation of detector response is provided by GEANT4 code. The standard multipurpose generators like Pythia6 and Pythia8 as well as specialised generators can be used for simulation of primary nucleon-nucleon collision.

What is FairMQ

What is FairMQ?

Organize processing tasks in topologies, consisting of independent processes (Device
that communicate via *asynchronous message queues* over network or inter-process.

Ethernet, InfiniBand (IP-over-IB)

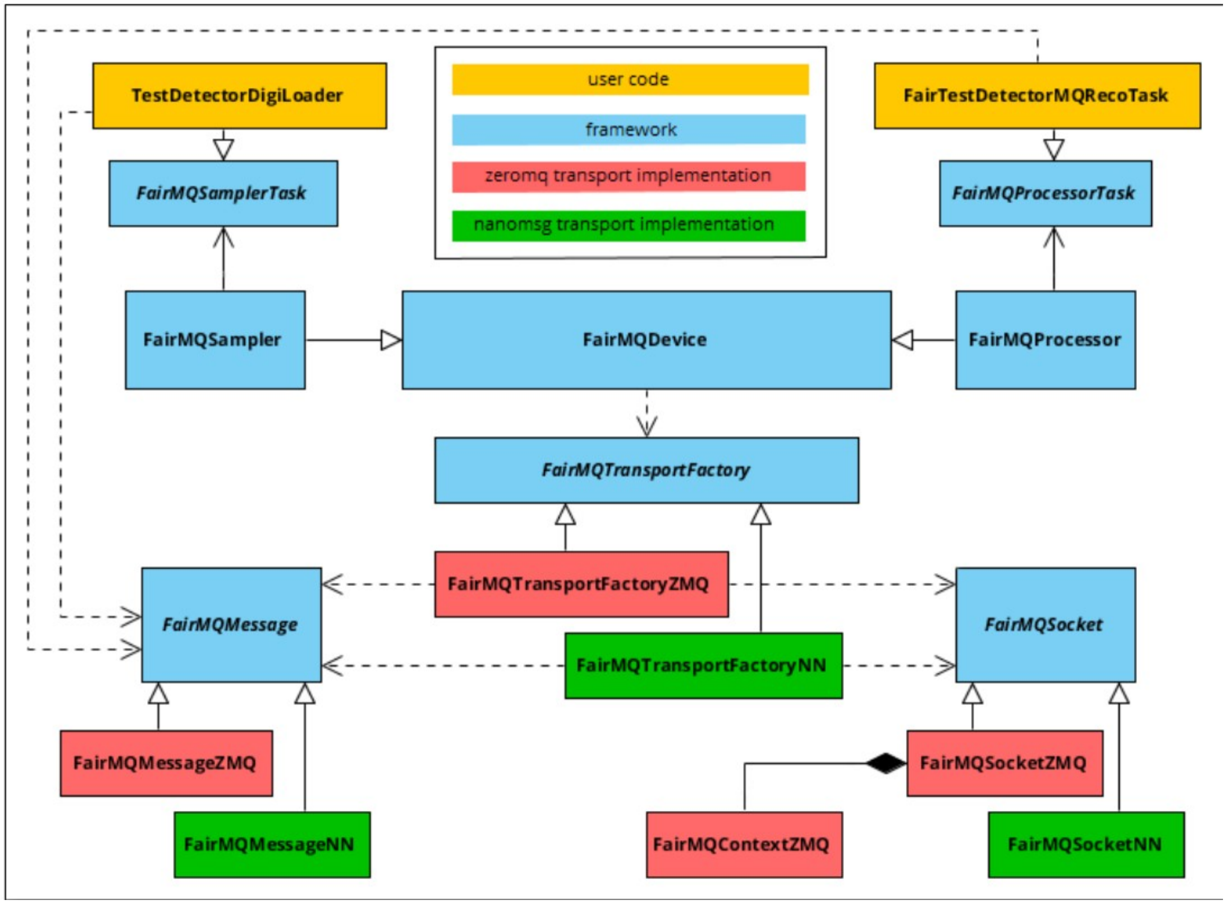

FAIRMQ TOPOLOGY (example)
SAMPLER
SAMPLER
SPLITTER
PROCESSOR
PROCESSOR
PROCESSOR
MERGER
SINK
Communication patterns: PUB-SUB, PUSH-PULL, REQ-REP, ...

Ready to use devices are provided for typical scenarios.
User-defined devices can be implemented by inheriting from `FairMQDevice`.

FairMQ structure

Transport Interface

FairMQ transport interface keeps the user code independent of the data transport implementation.

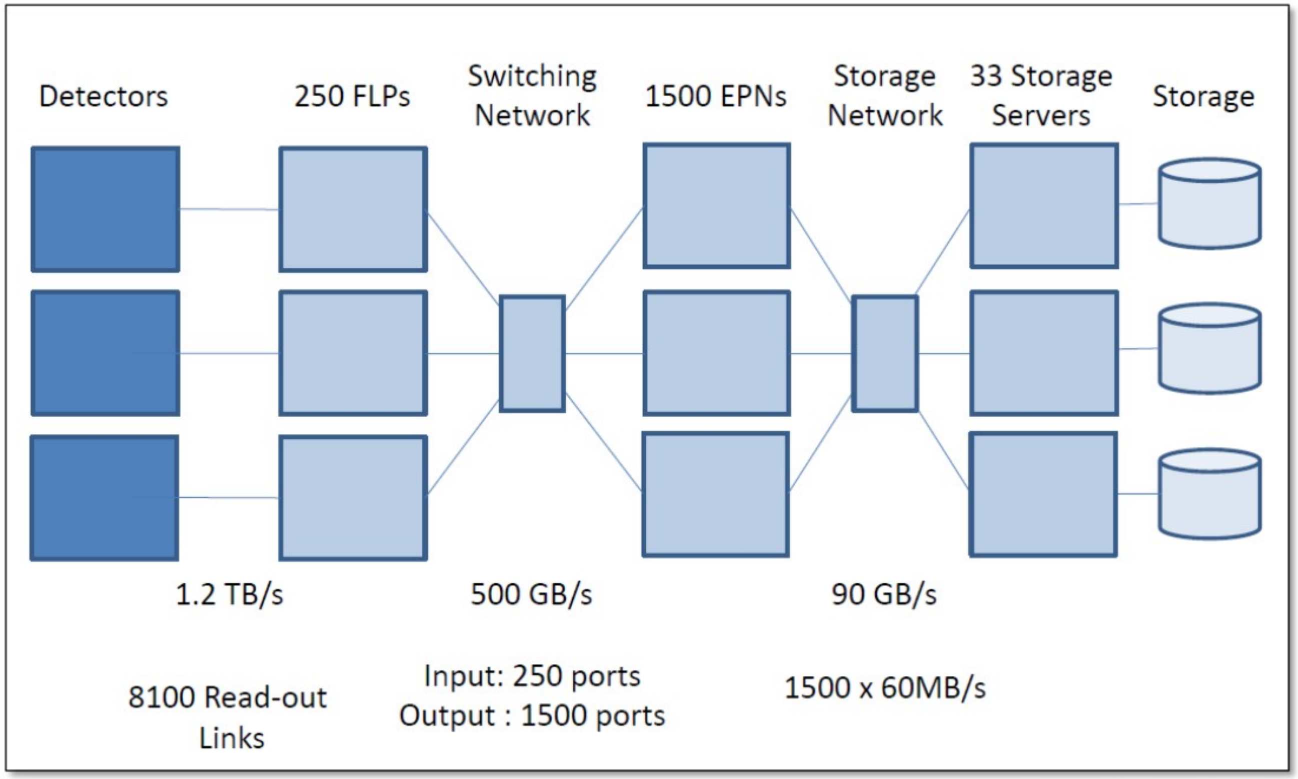


Currently two implementations:
With ZeroMQ or nanomsg libraries.

Possible implementation using future emerging technologies.

Experience of using on ALICE

Example Use Case: FLP2EPN Topology for ALICE O²



Source: O² Upgrade TDR

Parallelization throughput with FairMQ

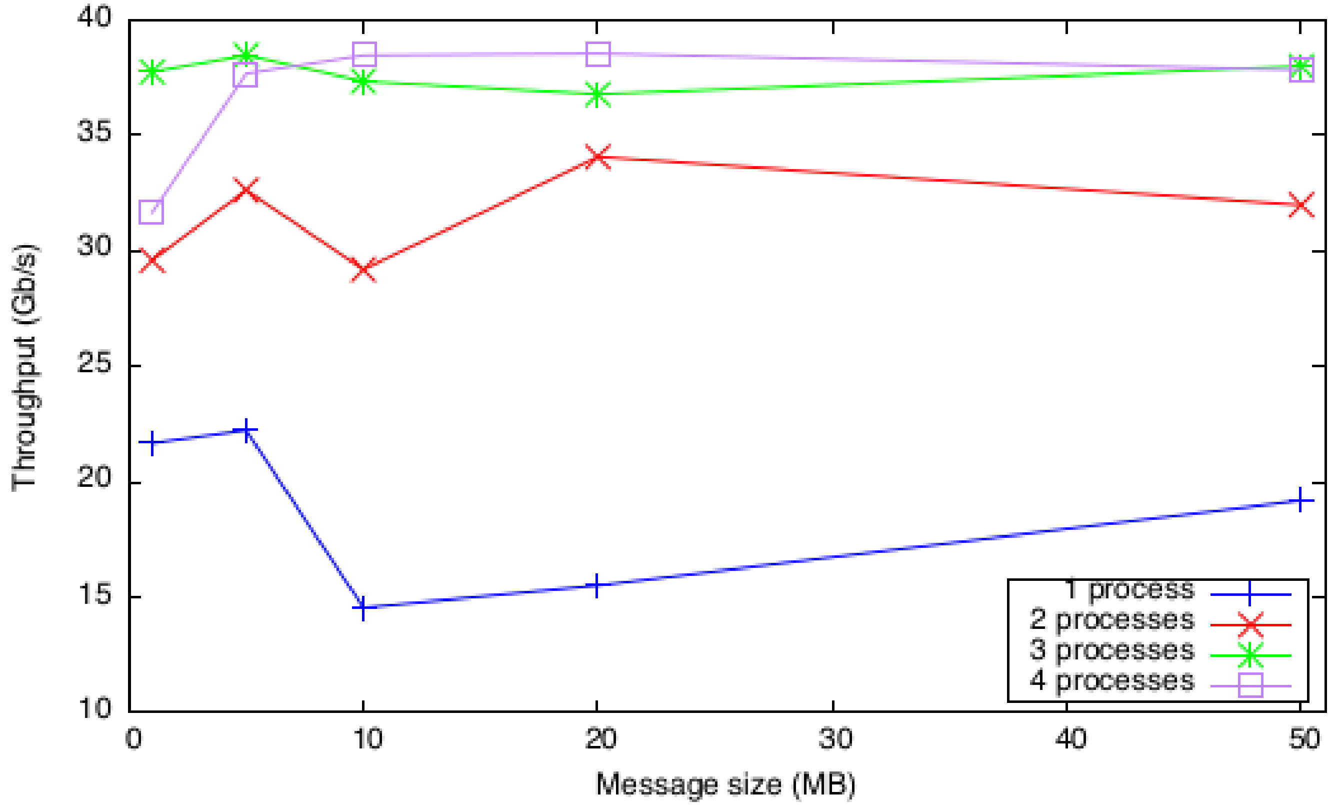


Figure 10.1: Throughput between two machines connected with 40 Gb/s Ethernet.

Data transport layer

The data transport layer is the part of the software which ensures the reliable arrival of message sand provides error checking mechanisms and data flow controls. The data transport layer in ALFA provides a number of components that can be connected to each other in order to construct a processing topology. They all share a common base class called device. Devices are grouped in three categories:

- Source: Devices without inputs are categorised as sources. A sampler is used to feed the pipeline (Task topology) with data from files.
- Message-based Processor: Devices that operate on messages without interpreting their content.
- Content-based Processor: This is the place where the message content is accessed and the user algorithms process the data.

Serialization

- Boost serialization. This method depends only on ANSI C++ facilities. Moreover, it exploits features of C++ such as RTTI (Run-Time Type Information), templates or multiple inheritance. It also provides independent versioning for each class definition. This means that when a class definition changes, older files can still be imported to the new version of the class. Another useful feature is the save and restore of deep pointers.
- Protocol buffers. Protocol buffers are Google’s language-neutral, platform-neutral, extensible mechanism for serializing structured data. The structure of the data is defined once and used to generate code to read and write data easily to and from a variety of data streams, using a variety of languages: Java, C++ or Python.
- ROOT. The ROOT Streamer can decompose ROOT objects into data members and write them to a buffer. This buffer can be written to a socket for sending over the network or to a file.
- User defined. In case it is decided not to use any of the above methods, binary structures or arrays can still be written or sent to a buffer. Although this method does not include any overhead for size of the data, issues can occur and will need to be managed. These include: schema evolution, different hardware, different languages.

DDS

The Dynamic Deployment System (DDS) is an independent set of utilities and interfaces, providing a dynamic distribution of different user processes for any given topology on any Resource Management System (RMS). The DDS uses a plug-in system in order to deploy different job submission front-ends. The first and the main plug-in of the system is a Secure Shell (SSH) that can be used to dynamically transform a set of machines into user worker nodes. The DDS functions are the following:

- Deploy a task or set of tasks
- Use any RMS (Slurm, Grid Engine, ...etc)
- Execute nodes securely (watchdog)
- Support different topologies and task dependencies
- Support a central log engine

During 2014, the core modules of the DDS were developed and the first stable prototype wasreleased. This has been tested on the ALICE HLT development cluster using 40 computingnodes with 32 processes per node. The SSH plugin for DDS has been used to successfullydistribute and manage 1281 ALICE O2user tasks (640 First Level Processor (FLPs) and 640Event Processing units (EPN)). The FLP processes here are emulating the FLP nodes whichwill collect the data wheres the EPN emulates the second step of data processing: assigningeach cluster to a track ([10])The DDS was able to propagate the allocated ports for each process to the dependentprocesses and set the required topology for the test. Throughout the test on this cluster, oneDDS commander server propagated more than 1.5 million properties in less than 5 seconds.

Performance measurement on ALICE O2

Two different systems were used for the performance measurement of data transport layer in ALFA. The performance tools delivered by ZeroMQ were also used to investigate any penalties introduced by FairMQ package. The goal of these measurement is to test the usability of the framework on different and existing system, so no effort was made to optimise or tune the network on the existing systems.

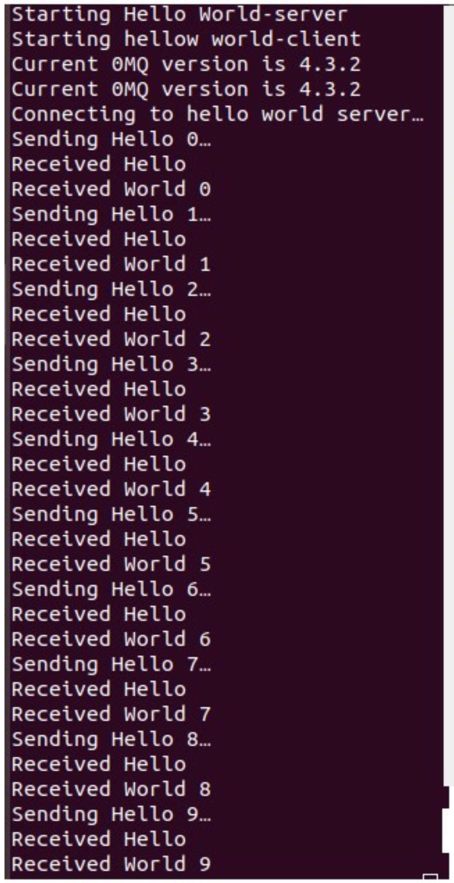
Ethernet-based prototype

This system consists of 8 dual-Xeon machines, 4 connected with 40 Gb Ethernet while the other 4 are connected with 10 Gb Ethernet. The throughput was measured as function of message size. For the ALICE RUN3 a message part size of 10 MB is expected, for this size,a rate of about 37.6 Gbs was achieved using 4 core CPUs for sending data between two of the machines (point to point). This test demonstrates that the overhead introduced by the FairMQ and ZeroMQ is marginal with a bandwidth equivalent to 94% of the theoretical one and that the technology scales well above the performance required by the FLPs on their output network link. More details are in Referance.

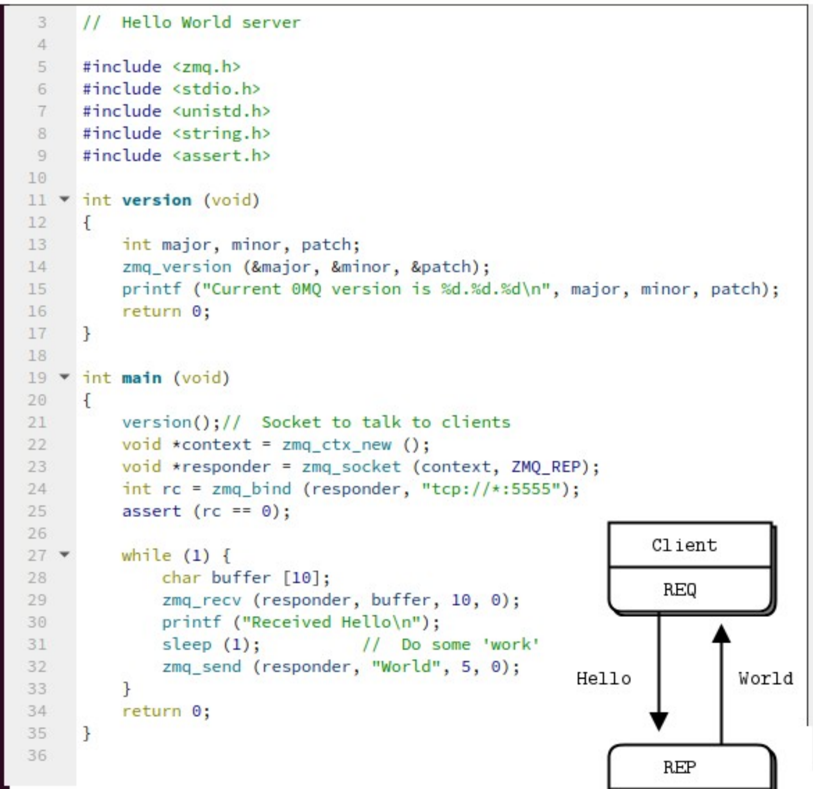
InfiniBand-based prototype

The second system is composed of a 40 Gb IB using 4 dual-Xeon machines (Intel Xeon E5520 with 4 physical cores and 8 threads each) all running the same software but with the IPoIBprotocol. Three processes were used to send data from one machine and 4 processes on each of the other machines received data. A message size of 10 MB was used. An average rate of2.5 GBs was reached without any optimisation of the kernel parameters. This test confirms that the marginal overhead introduced by the FairMQ and ZeroMQ software with a measured performance equivalent to the one measured with benchmarking programs. The test also demonstrates the portability of the FairMQ software to different network technologies (Ethernet and IB) which provides the independence about the underlying network technology.

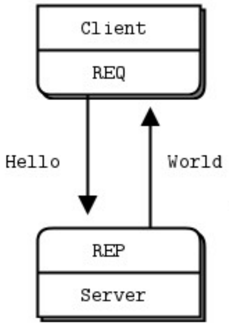
The simplest ZeroMQ pattern Request-Reply

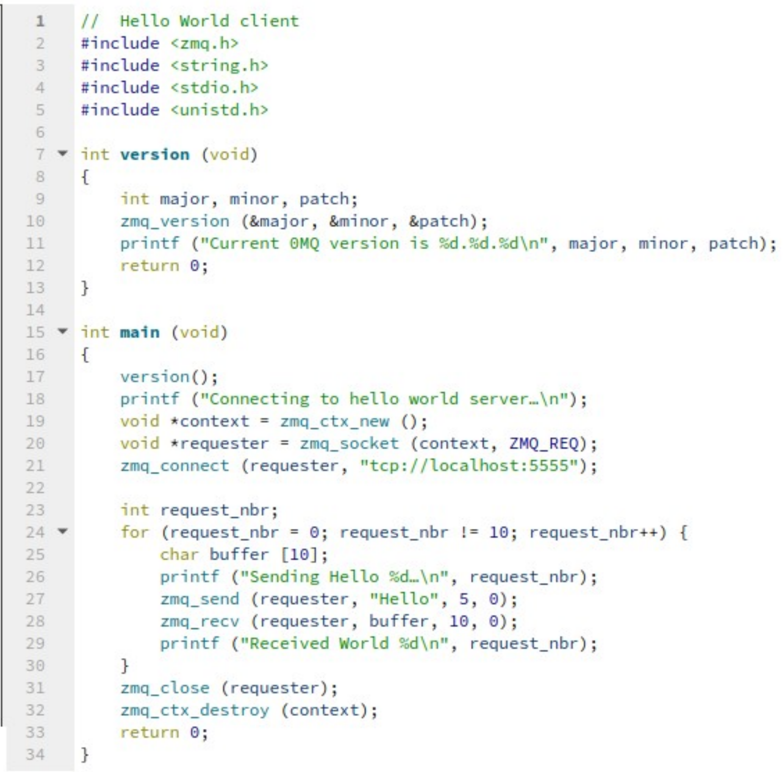


```
Starting Hello World-server
Starting hellow world-client
Current 0MQ version is 4.3.2
Current 0MQ version is 4.3.2
Connecting to hello world server…
Sending Hello 0…
Received Hello
Received World 0
Sending Hello 1…
Received Hello
Received World 1
Sending Hello 2…
Received Hello
Received World 2
Sending Hello 3…
Received Hello
Received World 3
Sending Hello 4…
Received Hello
Received World 4
Sending Hello 5…
Received Hello
Received World 5
Sending Hello 6…
Received Hello
Received World 6
Sending Hello 7…
Received Hello
Received World 7
Sending Hello 8…
Received Hello
Received World 8
Sending Hello 9…
Received Hello
Received World 9
```



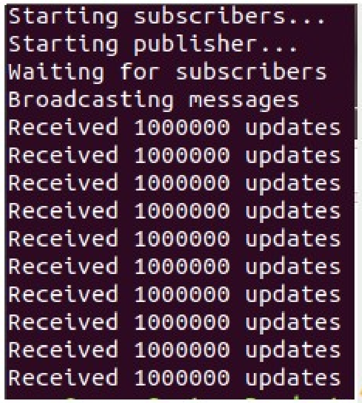
```c
//  Hello World server

#include <zmq.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>

int version (void)
{
    int major, minor, patch;
    zmq_version (&major, &minor, &patch);
    printf ("Current 0MQ version is %d.%d.%d\n", major, minor, patch);
    return 0;
}

int main (void)
{
    version();//  Socket to talk to clients
    void *context = zmq_ctx_new ();
    void *responder = zmq_socket (context, ZMQ_REP);
    int rc = zmq_bind (responder, "tcp://*:5555");
    assert (rc == 0);

    while (1) {
        char buffer [10];
        zmq_recv (responder, buffer, 10, 0);
        printf ("Received Hello\n");
        sleep (1);          //  Do some 'work'
        zmq_send (responder, "World", 5, 0);
    }
    return 0;
}
```



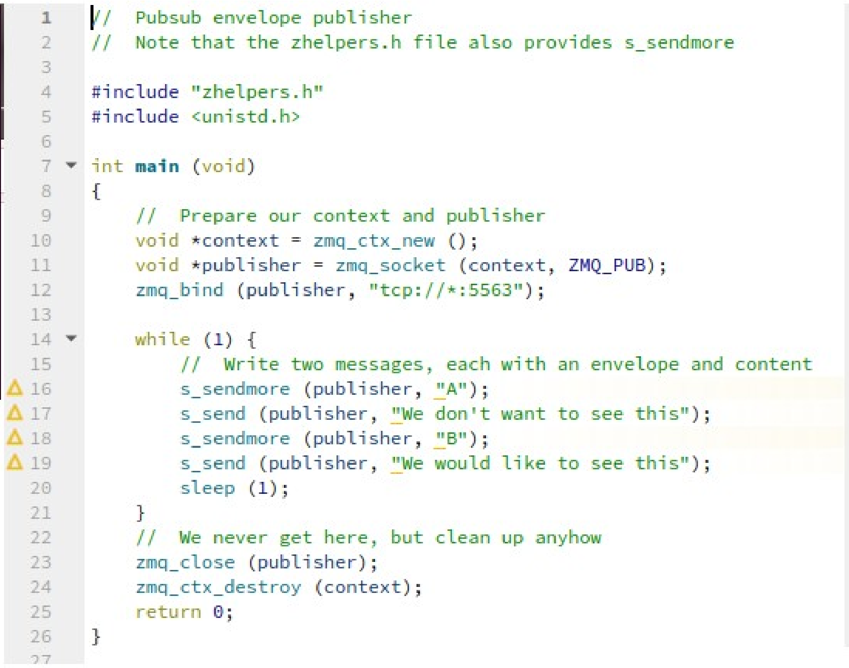


```c
//  Hello World client
#include <zmq.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>

int version (void)
{
    int major, minor, patch;
    zmq_version (&major, &minor, &patch);
    printf ("Current 0MQ version is %d.%d.%d\n", major, minor, patch);
    return 0;
}

int main (void)
{
    version();
    printf ("Connecting to hello world server…\n");
    void *context = zmq_ctx_new ();
    void *requester = zmq_socket (context, ZMQ_REQ);
    zmq_connect (requester, "tcp://localhost:5555");

    int request_nbr;
    for (request_nbr = 0; request_nbr != 10; request_nbr++) {
        char buffer [10];
        printf ("Sending Hello %d…\n", request_nbr);
        zmq_send (requester, "Hello", 5, 0);
        zmq_recv (requester, buffer, 10, 0);
        printf ("Received World %d\n", request_nbr);
    }
    zmq_close (requester);
    zmq_ctx_destroy (context);
    return 0;
}
```

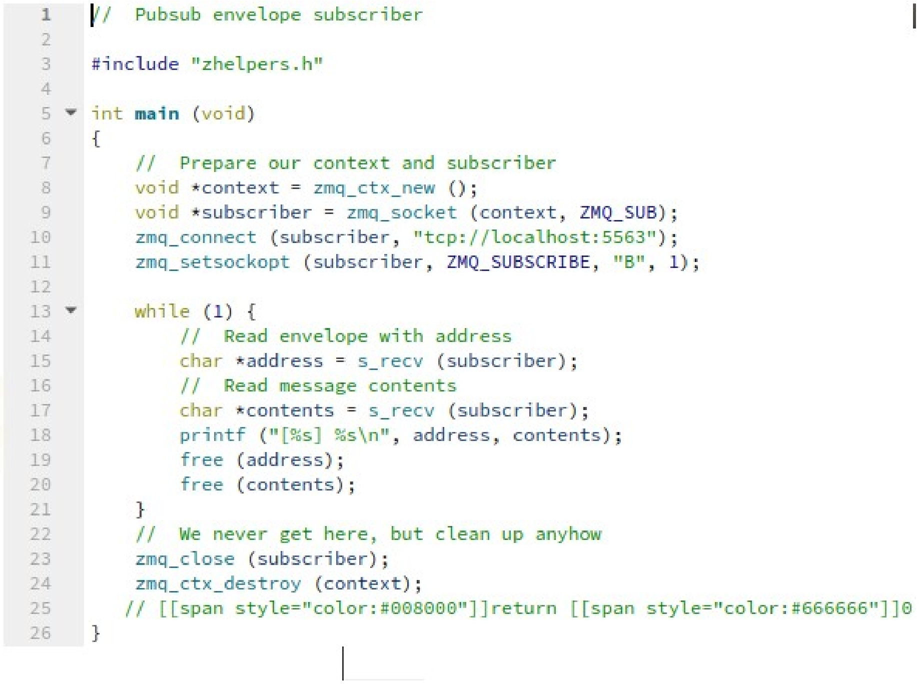
Publisher-Subscriber



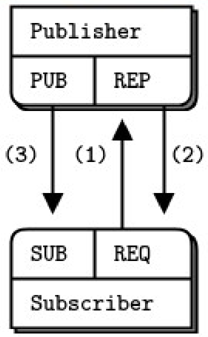
```
Starting subscribers...
Starting publisher...
Waiting for subscribers
Broadcasting messages
Received 1000000 updates
Received 1000000 updates
Received 1000000 updates
Received 1000000 updates
Received 1000000 updates
Received 1000000 updates
Received 1000000 updates
Received 1000000 updates
Received 1000000 updates
Received 1000000 updates
```



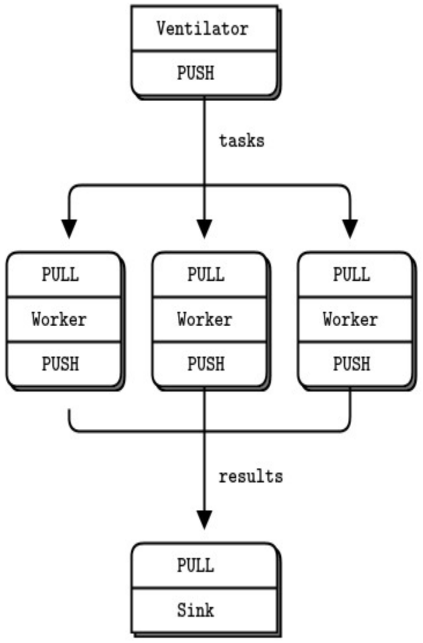
```c
//  Pubsub envelope publisher
//  Note that the zhelpers.h file also provides s_sendmore

#include "zhelpers.h"
#include <unistd.h>

int main (void)
{
    //  Prepare our context and publisher
    void *context = zmq_ctx_new ();
    void *publisher = zmq_socket (context, ZMQ_PUB);
    zmq_bind (publisher, "tcp://*:5563");

    while (1) {
        //  Write two messages, each with an envelope and content
        s_sendmore (publisher, "A");
        s_send (publisher, "We don't want to see this");
        s_sendmore (publisher, "B");
        s_send (publisher, "We would like to see this");
        sleep (1);
    }
    //  We never get here, but clean up anyhow
    zmq_close (publisher);
    zmq_ctx_destroy (context);
    return 0;
}
```



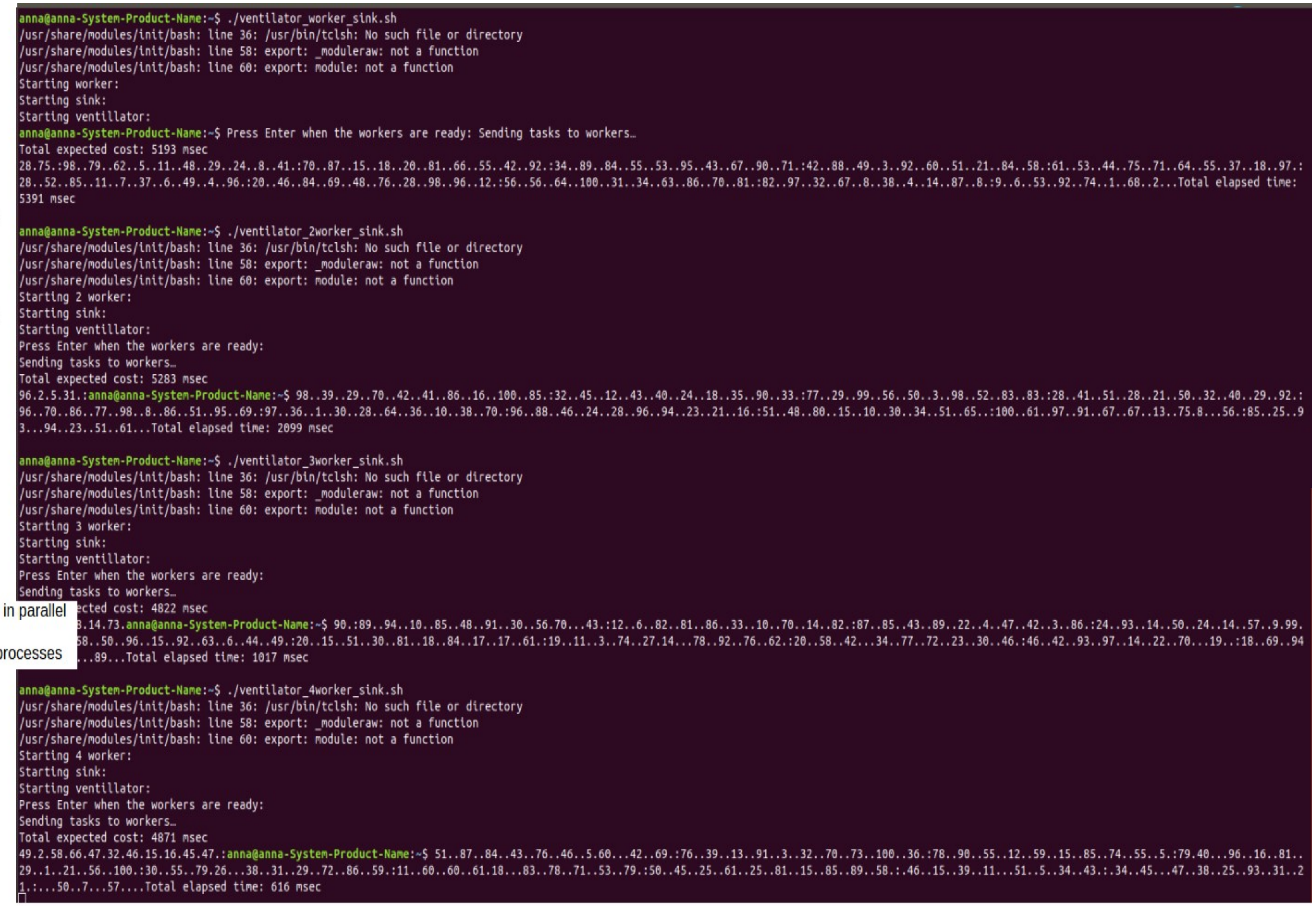
```c
//  Pubsub envelope subscriber

#include "zhelpers.h"

int main (void)
{
    //  Prepare our context and subscriber
    void *context = zmq_ctx_new ();
    void *subscriber = zmq_socket (context, ZMQ_SUB);
    zmq_connect (subscriber, "tcp://localhost:5563");
    zmq_setsockopt (subscriber, ZMQ_SUBSCRIBE, "B", 1);

    while (1) {
        //  Read envelope with address
        char *address = s_recv (subscriber);
        //  Read message contents
        char *contents = s_recv (subscriber);
        printf ("[%s] %s\n", address, contents);
        free (address);
        free (contents);
    }
    //  We never get here, but clean up anyhow
    zmq_close (subscriber);
    zmq_ctx_destroy (context);
   // [[span style="color:#008000"]]return [[span style="color:#666666"]]0
}
```



Ventillator-Worker-Sink

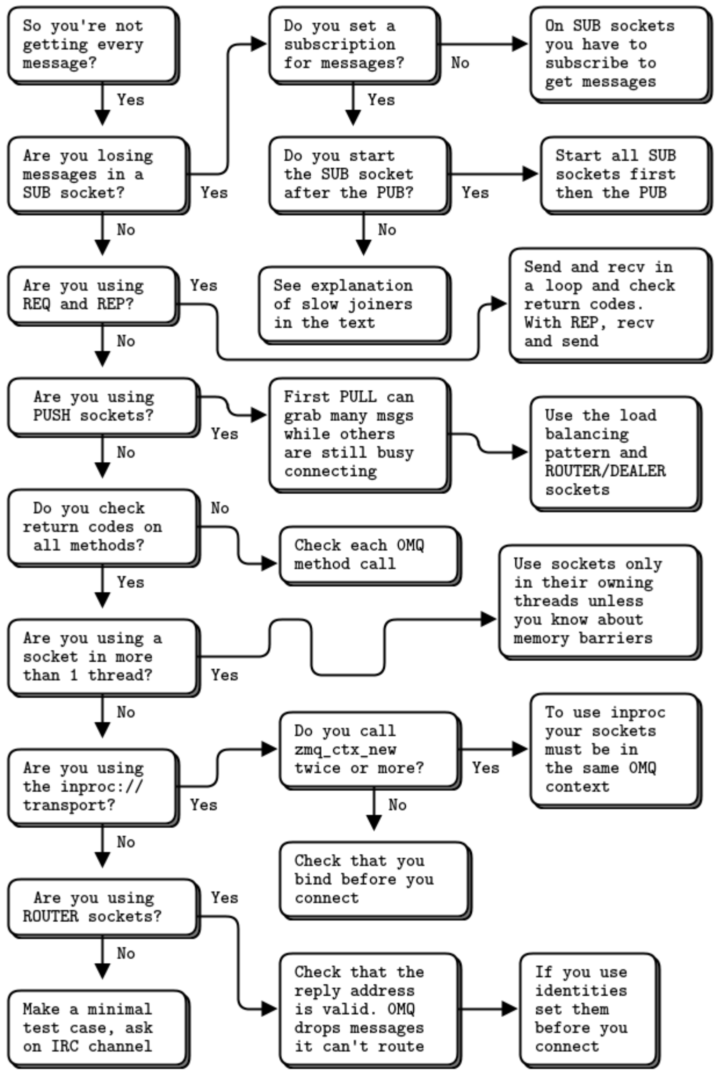


- A ventilator that produces tasks that can be done in parallel
- A set of workers that process tasks
- A sink that collects results back from the worker processes



```
anna@anna-System-Product-Name:~$ ./ventilator_worker_sink.sh
/usr/share/modules/init/bash: line 36: /usr/bin/tclsh: No such file or directory
/usr/share/modules/init/bash: line 58: export: _moduleraw: not a function
/usr/share/modules/init/bash: line 60: export: module: not a function
Starting worker:
Starting sink:
Starting ventillator:
anna@anna-System-Product-Name:~$ Press Enter when the workers are ready: Sending tasks to workers…
Total expected cost: 5193 msec
28.75.:98..79..62..5..11..48..29..24..8..41.:70..87..15..18..20..81..66..55..42..92.:34..89..84..55..53..95..43..67..90..71.:42..88..49..3..92..60..51..21..84..58.:61..53..44..75..71..64..55..37..18..97.:
28..52..85..11..7..37..6..49..4..96.:20..46..84..69..48..76..28..98..96..12.:56..56..64..100..31..34..63..86..70..81.:82..97..32..67..8..38..4..14..87..8.:9..6..53..92..74..1..68..2...Total elapsed time:
5391 msec

anna@anna-System-Product-Name:~$ ./ventilator_2worker_sink.sh
/usr/share/modules/init/bash: line 36: /usr/bin/tclsh: No such file or directory
/usr/share/modules/init/bash: line 58: export: _moduleraw: not a function
/usr/share/modules/init/bash: line 60: export: module: not a function
Starting 2 worker:
Starting sink:
Starting ventillator:
Press Enter when the workers are ready:
Sending tasks to workers…
Total expected cost: 5283 msec
96.2.5.31.:anna@anna-System-Product-Name:~$ 98..39..29..70..42..41..86..16..100..85.:32..45..12..43..40..24..18..35..90..33.:77..29..99..56..50..3..98..52..83..83.:28..41..51..28..21..50..32..40..29..92.:
96..70..86..77..98..8..86..51..95..69.:97..36..1..30..28..64..36..10..38..70.:96..88..46..24..28..96..94..23..21..16.:51..48..80..15..10..30..34..51..65..:100..61..97..91..67..67..13..75.8...56.:85..25..9
3...94..23..51..61...Total elapsed time: 2099 msec

anna@anna-System-Product-Name:~$ ./ventilator_3worker_sink.sh
/usr/share/modules/init/bash: line 36: /usr/bin/tclsh: No such file or directory
/usr/share/modules/init/bash: line 58: export: _moduleraw: not a function
/usr/share/modules/init/bash: line 60: export: module: not a function
Starting 3 worker:
Starting sink:
Starting ventillator:
Press Enter when the workers are ready:
Sending tasks to workers…
      ected cost: 4822 msec
      3.14.73.anna@anna-System-Product-Name:~$ 90.:89..94..10..85..48..91..30..56.70...43.:12..6..82..81..86..33..10..70..14..82.:87..85..43..89..22..4..47..42..3..86.:24..93..14..50..24..14..57..9.99.
      58..50..96..15..92..63..6..44..49.:20..15..51..30..81..18..84..17..17..61.:19..11..3..74..27.14...78..92..76..62.:20..58..42...34..77..72..23..30..46.:46..42..93..97..14..22..70...19..:18..69..94
      ...89...Total elapsed time: 1017 msec

anna@anna-System-Product-Name:~$ ./ventilator_4worker_sink.sh
/usr/share/modules/init/bash: line 36: /usr/bin/tclsh: No such file or directory
/usr/share/modules/init/bash: line 58: export: _moduleraw: not a function
/usr/share/modules/init/bash: line 60: export: module: not a function
Starting 4 worker:
Starting sink:
Starting ventillator:
Press Enter when the workers are ready:
Sending tasks to workers…
Total expected cost: 4871 msec
49.2.58.66.47.32.46.15.16.45.47.:anna@anna-System-Product-Name:~$ 51..87..84..43..76..46..5.60...42..69.:76..39..13..91..3..32..70..73..100..36.:78..90..55..12..59..15..85..74..55..5.:79.40...96..16..81..
29..1..21..56..100.:30..55..79.26...38..31..29..72..86..59.:11..60..60..61.18...83..78..71..53..79.:50..45..25..61..25..81..15..85..89..58.:.46..15..39..11...51..5..34..43.:.34..45...47..38..25..93..31..2
1.:...50..7...57....Total elapsed time: 616 msec
```

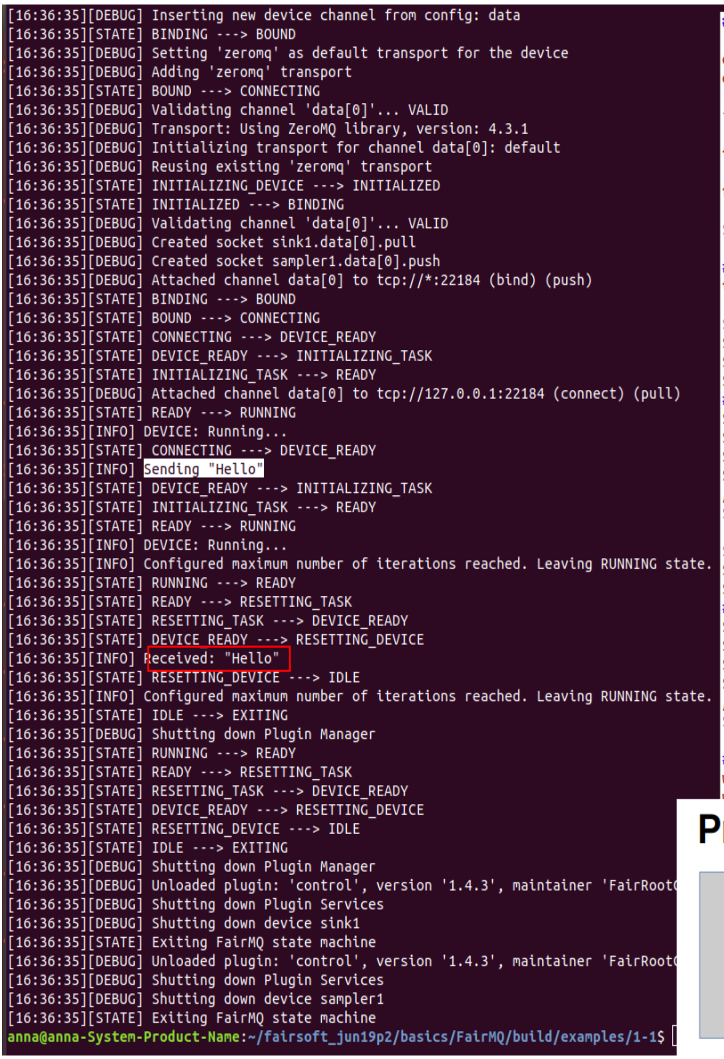
Problem solving in ZeroMQ


So you're not getting every message?
Yes
Are you losing messages in a SUB socket?
Yes
Do you set a subscription for messages?
No
On SUB sockets you have to subscribe to get messages
Yes
Do you start the SUB socket after the PUB?
Yes
Start all SUB sockets first then the PUB
No
See explanation of slow joiners in the text
No
Are you using REQ and REP?
Yes
Send and recv in a loop and check return codes. With REP, recv and send
No
Are you using PUSH sockets?
Yes
First PULL can grab many msgs while others are still busy connecting
Use the load balancing pattern and ROUTER/DEALER sockets
No
Do you check return codes on all methods?
No
Check each 0MQ method call
Yes
Are you using a socket in more than 1 thread?
Yes
Use sockets only in their owning threads unless you know about memory barriers
No
Are you using the inproc:// transport?
Yes
Do you call zmq_ctx_new twice or more?
Yes
To use inproc your sockets must be in the same 0MQ context
No
Check that you bind before you connect
No
Are you using ROUTER sockets?
Yes
Check that the reply address is valid. 0MQ drops messages it can't route
If you use identities set them before you connect
No
Make a minimal test case, ask on IRC channel

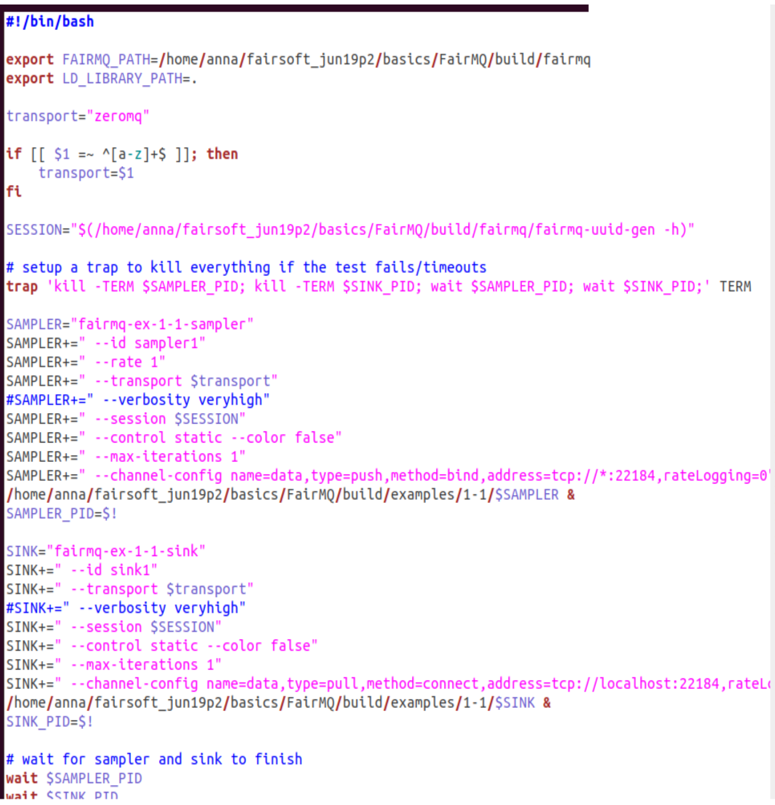
Benefits of using ZeroMQ

- The ability to create any custom data structures to be exchanged, ranging from an empty message to a full set of various characteristics
- When re-sending the same message, a "zero copy" occurs: a data link is sent. After sending any data, memory is freed
- Either the entire message is sent without errors, or nothing is sent, which prevents any data loss
- Possibility of excellent parallelization of processes
- Vertical and horizontal scaling
- When implemented with multithreading, the use of mutexes, lockers and symaphores is not required

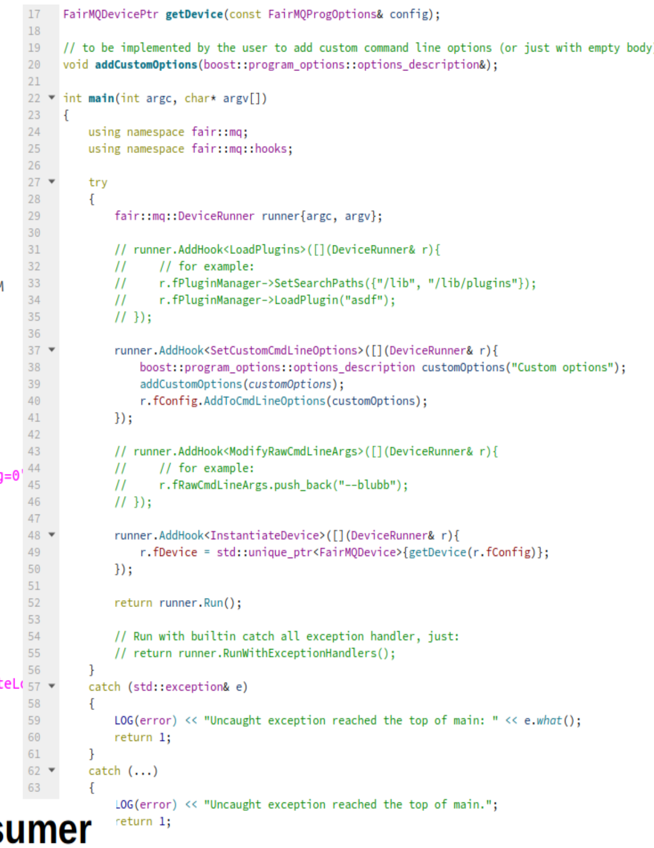
Producer-Consumer with FairMQ extending



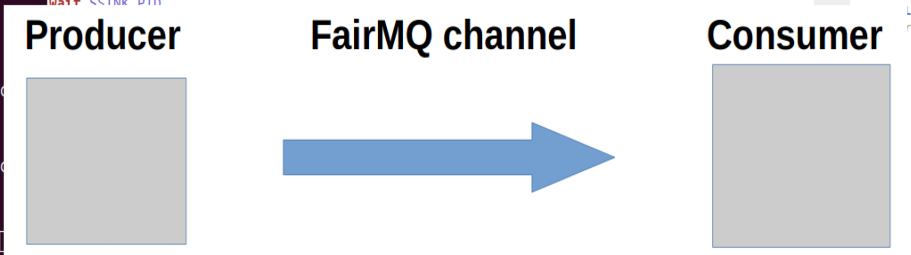
```
[16:36:35][DEBUG] Inserting new device channel from config: data
[16:36:35][STATE] BINDING ---> BOUND
[16:36:35][DEBUG] Setting 'zeromq' as default transport for the device
[16:36:35][DEBUG] Adding 'zeromq' transport
[16:36:35][STATE] BOUND ---> CONNECTING
[16:36:35][DEBUG] Validating channel 'data[0]'... VALID
[16:36:35][DEBUG] Transport: Using ZeroMQ library, version: 4.3.1
[16:36:35][DEBUG] Initializing transport for channel data[0]: default
[16:36:35][DEBUG] Reusing existing 'zeromq' transport
[16:36:35][STATE] INITIALIZING_DEVICE ---> INITIALIZED
[16:36:35][STATE] INITIALIZED ---> BINDING
[16:36:35][DEBUG] Validating channel 'data[0]'... VALID
[16:36:35][DEBUG] Created socket sink1.data[0].pull
[16:36:35][DEBUG] Created socket sampler1.data[0].push
[16:36:35][DEBUG] Attached channel data[0] to tcp://*:22184 (bind) (push)
[16:36:35][STATE] BINDING ---> BOUND
[16:36:35][STATE] BOUND ---> CONNECTING
[16:36:35][STATE] CONNECTING ---> DEVICE_READY
[16:36:35][STATE] DEVICE_READY ---> INITIALIZING_TASK
[16:36:35][STATE] INITIALIZING_TASK ---> READY
[16:36:35][DEBUG] Attached channel data[0] to tcp://127.0.0.1:22184 (connect) (pull)
[16:36:35][STATE] READY ---> RUNNING
[16:36:35][INFO] DEVICE: Running...
[16:36:35][STATE] CONNECTING ---> DEVICE_READY
[16:36:35][INFO] Sending "Hello"
[16:36:35][STATE] DEVICE_READY ---> INITIALIZING_TASK
[16:36:35][STATE] INITIALIZING_TASK ---> READY
[16:36:35][STATE] READY ---> RUNNING
[16:36:35][INFO] DEVICE: Running...
[16:36:35][INFO] Configured maximum number of iterations reached. Leaving RUNNING state.
[16:36:35][STATE] RUNNING ---> READY
[16:36:35][STATE] READY ---> RESETTING_TASK
[16:36:35][STATE] RESETTING_TASK ---> DEVICE_READY
[16:36:35][STATE] DEVICE_READY ---> RESETTING_DEVICE
[16:36:35][INFO] Received: "Hello"
[16:36:35][STATE] RESETTING_DEVICE ---> IDLE
[16:36:35][INFO] Configured maximum number of iterations reached. Leaving RUNNING state.
[16:36:35][STATE] IDLE ---> EXITING
[16:36:35][DEBUG] Shutting down Plugin Manager
[16:36:35][STATE] RUNNING ---> READY
[16:36:35][STATE] READY ---> RESETTING_TASK
[16:36:35][STATE] RESETTING_TASK ---> DEVICE_READY
[16:36:35][STATE] DEVICE_READY ---> RESETTING_DEVICE
[16:36:35][STATE] RESETTING_DEVICE ---> IDLE
[16:36:35][STATE] IDLE ---> EXITING
[16:36:35][DEBUG] Shutting down Plugin Manager
[16:36:35][DEBUG] Unloaded plugin: 'control', version '1.4.3', maintainer 'FairRoot
[16:36:35][DEBUG] Shutting down Plugin Services
[16:36:35][DEBUG] Shutting down device sink1
[16:36:35][STATE] Exiting FairMQ state machine
[16:36:35][DEBUG] Unloaded plugin: 'control', version '1.4.3', maintainer 'FairRoot
[16:36:35][DEBUG] Shutting down Plugin Services
[16:36:35][DEBUG] Shutting down device sampler1
[16:36:35][STATE] Exiting FairMQ state machine
anna@anna-System-Product-Name:~/fairsoft_jun19p2/basics/FairMQ/build/examples/1-1$
```



```bash
#!/bin/bash

export FAIRMQ_PATH=/home/anna/fairsoft_jun19p2/basics/FairMQ/build/fairmq
export LD_LIBRARY_PATH=.

transport="zeromq"

if [[ $1 =~ ^[a-z]+$ ]]; then
    transport=$1
fi

SESSION="$(/home/anna/fairsoft_jun19p2/basics/FairMQ/build/fairmq/fairmq-uuid-gen -h)"

# setup a trap to kill everything if the test fails/timeouts
trap 'kill -TERM $SAMPLER_PID; kill -TERM $SINK_PID; wait $SAMPLER_PID; wait $SINK_PID;' TERM

SAMPLER="fairmq-ex-1-1-sampler"
SAMPLER+=" --id sampler1"
SAMPLER+=" --rate 1"
SAMPLER+=" --transport $transport"
#SAMPLER+=" --verbosity veryhigh"
SAMPLER+=" --session $SESSION"
SAMPLER+=" --control static --color false"
SAMPLER+=" --max-iterations 1"
SAMPLER+=" --channel-config name=data,type=push,method=bind,address=tcp://*:22184,rateLogging=0'
/home/anna/fairsoft_jun19p2/basics/FairMQ/build/examples/1-1/$SAMPLER &
SAMPLER_PID=$!

SINK="fairmq-ex-1-1-sink"
SINK+=" --id sink1"
SINK+=" --transport $transport"
#SINK+=" --verbosity veryhigh"
SINK+=" --session $SESSION"
SINK+=" --control static --color false"
SINK+=" --max-iterations 1"
SINK+=" --channel-config name=data,type=pull,method=connect,address=tcp://localhost:22184,rateL
/home/anna/fairsoft_jun19p2/basics/FairMQ/build/examples/1-1/$SINK &
SINK_PID=$!

# wait for sampler and sink to finish
wait $SAMPLER_PID
wait $SINK_PID
```



```cpp
FairMQDevicePtr getDevice(const FairMQProgOptions& config);

// to be implemented by the user to add custom command line options (or just with empty body
void addCustomOptions(boost::program_options::options_description&);

int main(int argc, char* argv[])
{
    using namespace fair::mq;
    using namespace fair::mq::hooks;

    try
    {
        fair::mq::DeviceRunner runner{argc, argv};

        // runner.AddHook<LoadPlugins>([](DeviceRunner& r){
        //     // for example:
        //     r.fPluginManager->SetSearchPaths({"/lib", "/lib/plugins"});
        //     r.fPluginManager->LoadPlugin("asdf");
        // });

        runner.AddHook<SetCustomCmdLineOptions>([](DeviceRunner& r){
            boost::program_options::options_description customOptions("Custom options");
            addCustomOptions(customOptions);
            r.fConfig.AddToCmdLineOptions(customOptions);
        });

        // runner.AddHook<ModifyRawCmdLineArgs>([](DeviceRunner& r){
        //     // for example:
        //     r.fRawCmdLineArgs.push_back("--blubb");
        // });

        runner.AddHook<InstantiateDevice>([](DeviceRunner& r){
            r.fDevice = std::unique_ptr<FairMQDevice>{getDevice(r.fConfig)};
        });

        return runner.Run();

        // Run with builtin catch all exception handler, just:
        // return runner.RunWithExceptionHandlers();
    }
    catch (std::exception& e)
    {
        LOG(error) << "Uncaught exception reached the top of main: " << e.what();
        return 1;
    }
    catch (...)
    {
        LOG(error) << "Uncaught exception reached the top of main.";
        return 1;
```



Conclusions

FairMQ uses ZeroMQ as its main transport layout and therefore has superior process parallelization, data integrity, and easy multithreading capabilities. ALICE O2 experiments have demonstrated high throughput using FairMQ, and therefore, there are good prospects for using the FairMQ package in SPD experiments.

References

1. http://spd.jinr.ru/spd-software/

2. Alexey Rybalchenko, GSI Darmstadt, FairRoot group, FairMQ Data Transport for Online & Offline Processing, ALICE Offline Week CERN, July 1, 2015

3. M. Al-Turany1,2, P. Buncic2, P. Hristov2, T. Kollegger1, C.Kouzinopoulos2, A. Lebedev1, V. Lindenstruth1,3, A. Manafov1, M.Richter2,4, A. Rybalchenko1, P. Vande Vyvre2, N. Winckler: ALFA: The new ALICE-FAIR software framework

4. http://wiki.zeromq.org/intro:read-the-manual

The end

Thank you for attention!