Fabio Galuppo, M.Sc.

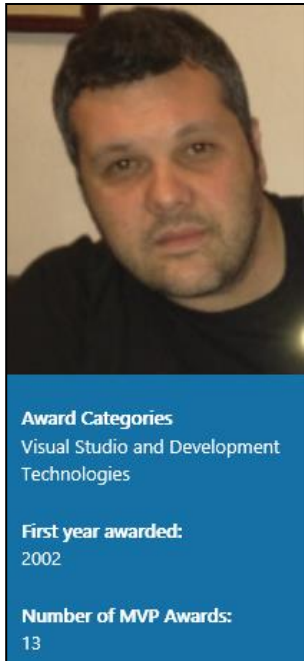
http://fabiogaluppo.com e http://simplycpp.com/

fabiogaluppo@acm.org

@FabioGaluppo

Microsoft MVP Visual Studio and Development Technologies

https://mvp.microsoft.com/en-us/PublicProfile/9529



O que é ZeroMQ?

- *Intelligent socket library for messaging*
  - Variedade nos padrões de comunicação
    - Request-Reply, Publisher-Subscriber, Push-Pull, Dealer-Router, …
  - Suporta: *inproc*, *IPC*, *TCP*, *TIPC*, *multicast*
- Modelo de concorrência baseado em atores (*Erlang-style)*
- *Open Source*
- Multiplas plataformas
- Diversas linguagens (mais de 30)
  - C, C++, Java, C#, Python, …
- *Deploy* simples (uma *library*)
- Alta performance
  - http://zeromq.org/results:multicore-tests
    - ~6 milhões de mensagens por segundo

Destaque

Who is Using ZeroMQ?

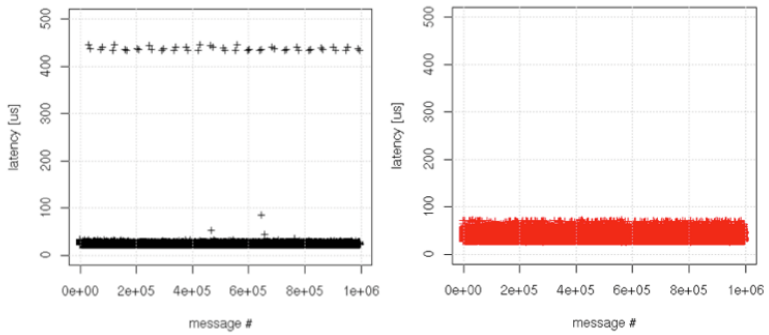
Since ZeroMQ is free software we don't track who uses it. However, some organizations that we know use it are: AT&T, Cisco, EA, Los Alamos Labs, NASA, Weta Digital, Zynga, Spotify, Samsung Electronics, Microsoft, and CERN.

- Cisco: The Avalanche Project: When High Frequency Trading Meets Traffic Classification
- CERN: MIDDLEWARE TRENDS AND MARKET LEADERS 2011

http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/frbhmult05.pdf

CHOOSING A ROBUST MESSAGING SYSTEM

The foundation of our architecture relies on choosing the right messaging library. This will be a key factor to determine how well our data processing pipeline will perform. I will save you from all the details of a descriptive comparison between all the technology, but ZeroMQ is one of the best messaging middleware available, and it is very well known in the finance world. It also provides an amazing paradigm to build a distributed system with different message passing patterns. During my analysis, some important metrics caught my attention :



Source: http://zeromq.org/results:rt-tests-v031

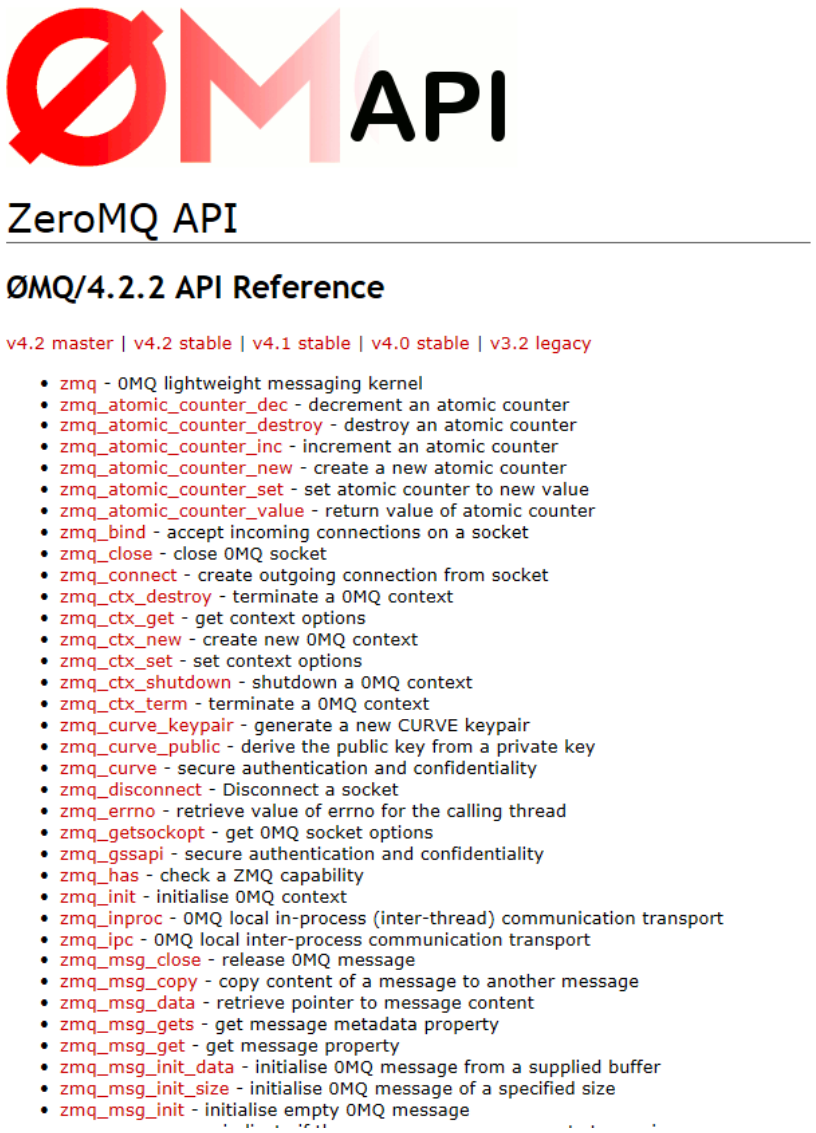
	patterns	QoS	resources	performance	user friendly	community	score
CORBA	✗	✓	✗	✓	✗	✗	2
Ice	✓	✓	✗	✓	✓	✓	5
Thrift	✗	✗	✓	✓	✗	✗	2
ZeroMQ	✓	✓	✓	✓	✓	✓	6
YAMI4	✓	✓	✓	✗	✓	✗	4
RTI	✗	✓	✗	✓	✗	✓	3
Qpid	✗	✓	✗	✗	✓	✓	3

Figure 3: Summary of evaluated middleware products.

https://umbrella.cisco.com/blog/2015/11/05/the-avalanche-project-when-high-frequency-trading-meets-traffic-classification/

ZeroMQ: APIs essenciais

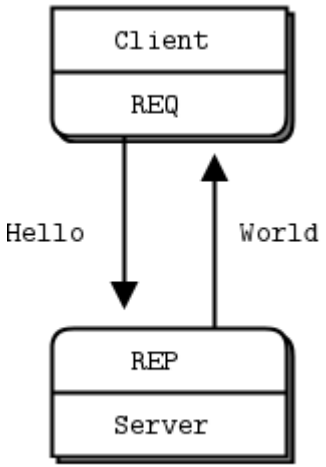
- http://api.zeromq.org/
  - zmq_ctx_new - create new 0MQ context
  - zmq_term - terminate 0MQ context
  - zmq_socket - create 0MQ socket
  - zmq_close - close 0MQ socket
  - zmq_bind - accept incoming connections on a socket
  - zmq_connect - create outgoing connection from socket
  - zmq_send - send a message part on a socket
    - zmq_send, zmq_sendmsg, zmq_send_const
  - zmq_recv - receive a message part from a socket
    - zmq_recv, zmq_recvmsg
  - zmq_setsockopt - set 0MQ socket options
  - zmq_getsockopt - get 0MQ socket options



Request-Reply Pattern

```c
void* zmq_context = zmq_ctx_new();
void* request_socket = zmq_socket(zmq_context, ZMQ_REQ);
zmq_connect(request_socket, "tcp://localhost:60000");

char* msg = "Hello, World!";
if (argc > 1)
    msg = argv[1];

zmq_send(request_socket, msg, strlen(msg), 0);

char buffer[64];
zmq_recv(request_socket, buffer, sizeof(buffer), 0);
printf("%s\n", buffer);

zmq_close(request_socket);
zmq_ctx_term(zmq_context);
```

```c
void* zmq_context = zmq_ctx_new();
void* reply_socket = zmq_socket(zmq_context, ZMQ_REP);
if (0 == zmq_bind(reply_socket, "tcp://*:60000")) {
    char buffer[64];
    zmq_recv(reply_socket, buffer, sizeof(buffer), 0);
    printf("%s\n", buffer);
    size_t N = strlen(buffer);
    for (size_t i = 0; i < N; ++i)
        buffer[i] = toupper(buffer[i]);
    zmq_send(reply_socket, buffer, N, 0);
}
else {
    printf("%s\n", strerror(errno));
}

zmq_close(reply_socket);
zmq_ctx_term(zmq_context);
```




```
sample-1 — bash — 80×24
Fabios-MacBook-Pro:sample-1 fabiogaluppo$ clang -I../include -lzmq req.c -
n/req.exe
Fabios-MacBook-Pro:sample-1 fabiogaluppo$ ./bin/req.exe
ZMQ version = 4.2.3
HELLO, WORLD!
Fabios-MacBook-Pro:sample-1 fabiogaluppo$ ./bin/req.exe "Testing 1, 2, 3"
ZMQ version = 4.2.3
TESTING 1, 2, 3
Fabios-MacBook-Pro:sample-1 fabiogaluppo$
```

```
sample-1 — bash — 80×24
Fabios-MacBook-Pro:sample-1 fabiogaluppo$ clang -I../incl
n/rep.exe
Fabios-MacBook-Pro:sample-1 fabiogaluppo$ ./bin/rep.exe
ZMQ version = 4.2.3
Hello, World!
Fabios-MacBook-Pro:sample-1 fabiogaluppo$ ./bin/rep.exe
ZMQ version = 4.2.3
Testing 1, 2, 3
Fabios-MacBook-Pro:sample-1 fabiogaluppo$
```

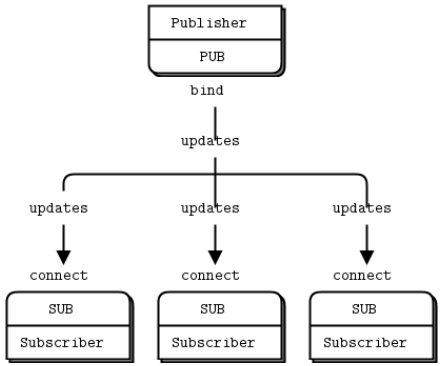
Publisher-Subscriber Pattern

```cpp
zmq::context_t zmq_context(1);
zmq::socket_t publisher_socket(zmq_context, ZMQ_PUB);

publisher_socket.bind("tcp://*:60001");
publisher_socket.bind("ipc://pub.ipc");

std::cout << "Publishing...\n";
std::cout << "[CTRL + C] to finish...\n";

std::string data;
while (std::getline(std::cin, data)) {
    std::transform(data.begin(), data.end(), data.begin(),
        [](int ch){ return std::toupper(ch); });
    publisher_socket.send(zmq::message_t(data.begin(), data.end()), 0);
}

publisher_socket.close();
zmq_context.close();
```

Publisher - C++ Binding (cppzmq) – 2 *endpoints* (tcp, ipc)

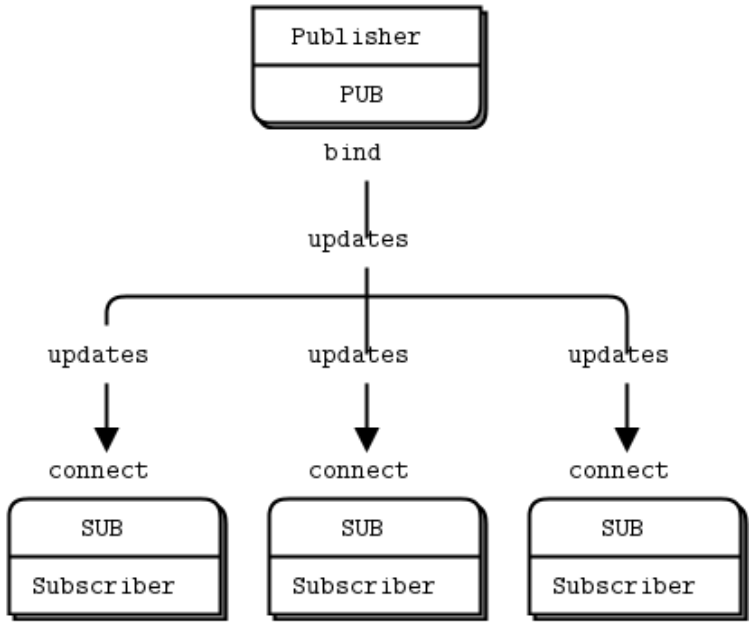
```cpp
zmq::context_t zmq_context(1);
zmq::socket_t subscriber_socket(zmq_context, ZMQ_SUB);

subscriber_socket.connect("tcp://localhost:60001");

std::cout << "Listening...\n";
std::cout << "[CTRL + C] to finish...\n";

for (int i = 2; i < argc; ++i) {
    const char* filter = argv[i];
    subscriber_socket.setsockopt(ZMQ_SUBSCRIBE, filter, std::strlen(filter));
}

for (long i = 0; i < N; ++i) {
    zmq::message_t msg;
    subscriber_socket.recv(&msg, 0);

    std::cout << to_string(msg) << "\n";
}

subscriber_socket.close();
zmq_context.close();
```

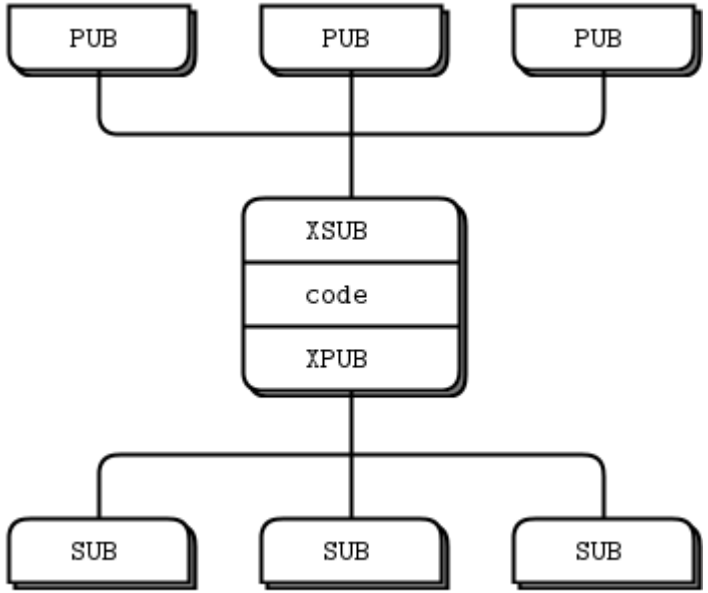
Subscriber – C++ Binding (cppzmq)



Publisher-Subscriber Pattern

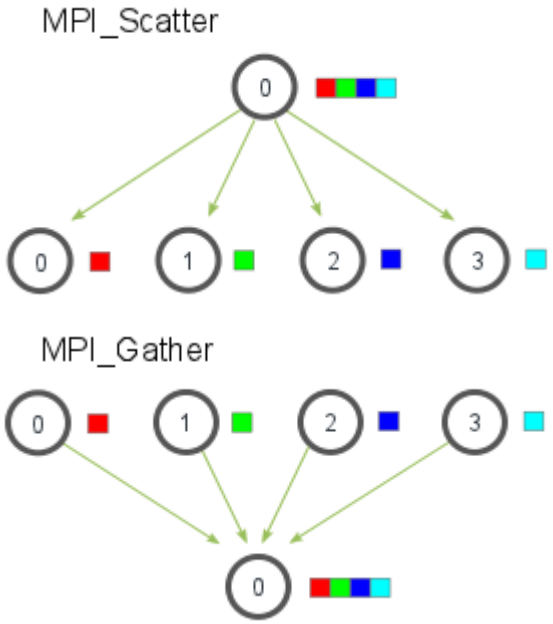
- PUB-SUB versus XPUB-XSUB



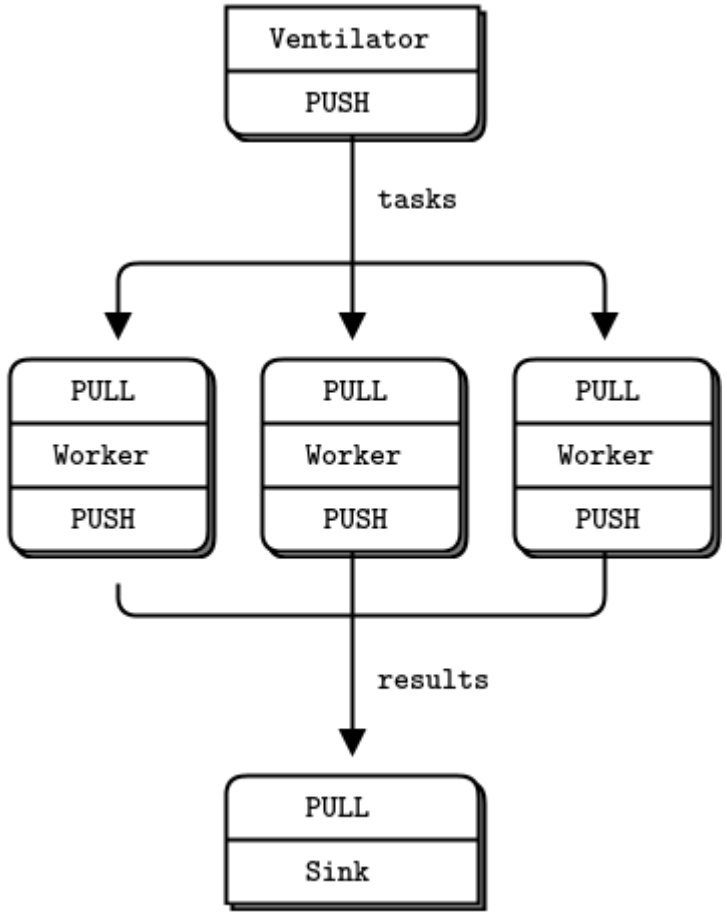


Parallel Pipeline Pattern

- Divide and Conquer
  - Fork-Join
  - Map-Reduce
  - MPI_Scatter-MPI_Gather



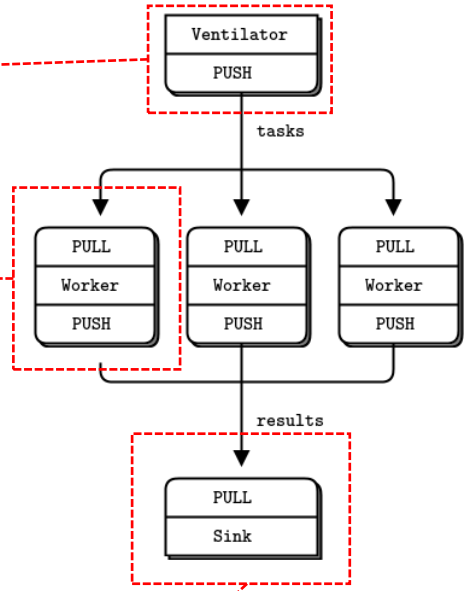
http://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/



Parallel Pipeline Pattern

```cpp
zmq::context_t zmq_context(2);
zmq::socket_t source_socket(zmq_context, ZMQ_PUSH);
zmq::socket_t sink_socket(zmq_context, ZMQ_PUSH);

zmq_utils::set_affinity(source_socket, 1);
zmq_utils::set_affinity(sink_socket, 2);

source_socket.bind("tcp://*:60002");
sink_socket.connect("tcp://localhost:60003");

//notify # of messages to sink
std::string SM = std::to_string(M);
sink_socket.send(zmq::message_t(SM.begin(), SM.end()), 0);

std::cout << "Waiting for workers...\n";
std::cout << "Press [ENTER] when workers are ready...\n";
std::getchar();
std::cout << "Publishing...\n";

seed_rand();
for (long i = 0; i < M; ++i) {
    int coin = rand_int(1, 2);
    if (coin == 1) {
        source_socket.send(command_msg(COMMAND_1), ZMQ_SNDMORE);
        source_socket.send(command_1_args(i + 1, i + 1), 0);
    }
    else {
        source_socket.send(command_msg(COMMAND_2), ZMQ_SNDMORE);
        char a = rand_int<short>(97, 122);
        bool b = rand_int(1, 2) == 1;
        char c = rand_int<short>(97, 122);
        source_socket.send(command_2_args(a, b, c), 0);
    }
}

for (long i = 0; i < N; ++i) {
    source_socket.send(command_msg(POISON_PILL), 0);
```

```cpp
zmq::context_t zmq_context(2);
zmq::socket_t worker_socket(zmq_context, ZMQ_PULL);
zmq::socket_t sink_socket(zmq_context, ZMQ_PUSH);

zmq_utils::set_affinity(worker_socket, 1);
zmq_utils::set_affinity(sink_socket, 2);

worker_socket.connect("tcp://localhost:60002");
sink_socket.connect("tcp://localhost:60003");

while (true) {
    std::cout << "Waiting message...\n";

    zmq::message_t msg;
    worker_socket.recv(&msg, 0);
    auto scmd = zmq_utils::to_string(msg);
    if (is_command(scmd, POISON_PILL))
        break;

    std::cout << "Processing message...\n";

    worker_socket.recv(&msg, 0);
    auto result = process(scmd, msg);

    std::cout << "Sending result...\n";
    sink_socket.send(result, 0);

    std::cout << "-----------------\n";
}
```

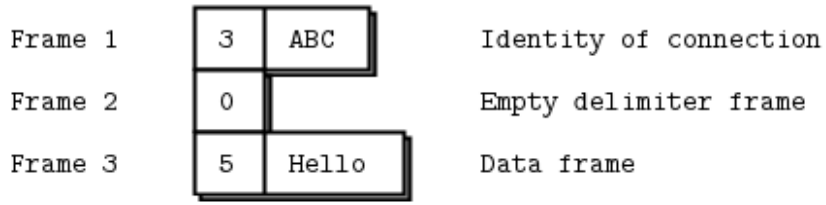



```cpp
zmq::context_t zmq_context(1);
zmq::socket_t sink_socket(zmq_context, ZMQ_PULL);

sink_socket.bind("tcp://*:60003");

std::cout << "Waiting for start...\n";

zmq::message_t msg;
sink_socket.recv(&msg, 0);
int M = std::stoi(zmq_utils::to_string(msg));
std::cout << "Waiting " << M << " messages...\n";
for (int i = 0; i < M; ++i) {
    sink_socket.recv(&msg, 0);
    std::cout << "Result #" << (i + 1) << ": " << zmq_ut
}
```

Multipart-Messages e Variadic Templates

- http://zguide.zeromq.org/page:all#Multipart-Messages

```
zmq_msg_send (&message, socket, ZMQ_SNDMORE);
…
zmq_msg_send (&message, socket, ZMQ_SNDMORE);
…
zmq_msg_send (&message, socket, 0);
```




```
template <typename... Ts>
inline result_type bulk_send(zmq::socket_t& sender_socket, Ts&... args) {
    size_t total_bytes_sent = 0;
    bool success = true;
    internal::variadic::bulk_sender<Ts...>::send(sender_socket, total_bytes_sent, success, args...);
    return std::make_tuple(success, total_bytes_sent);
}
```

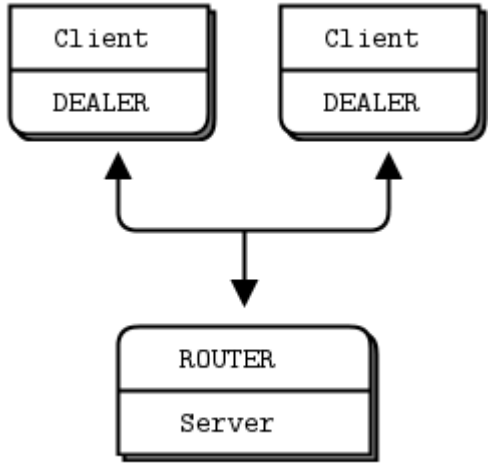
```
zmq::message_t data_1_msg { data1.begin(), data1.end() };
zmq::message_t data_2_msg { data2.begin(), data2.end() };

bulk_send(dealer, data_1_msg, data_2_msg);
```

Dealer-Router Pattern

Now we can switch out both REQ and REP with DEALER and ROUTER to get the most powerful socket combination, which is DEALER talking to ROUTER. It gives us asynchronous clients talking to asynchronous servers, where both sides have full control over the message formats.

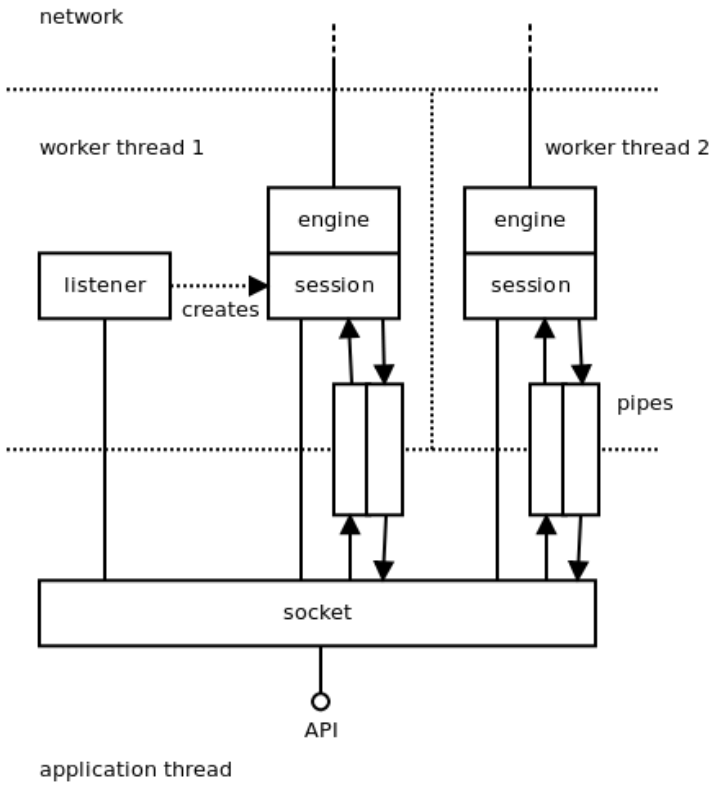
Because both DEALER and ROUTER can work with arbitrary message formats, if you hope to use these safely, you have to become a little bit of a protocol designer. At the very least you must decide whether you wish to emulate the REQ/REP reply envelope. It depends on whether you actually need to send replies or not.

http://zguide.zeromq.org/page:all#The-DEALER-to-ROUTER-Combination

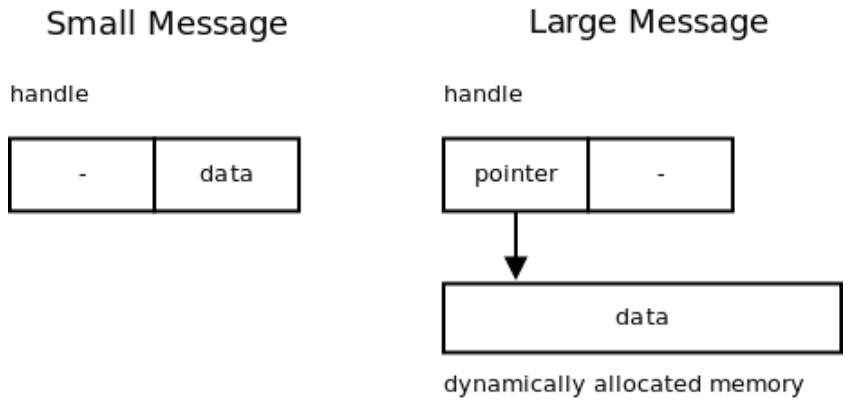


http://zguide.zeromq.org/page:all#The-Asynchronous-Client-Server-Pattern

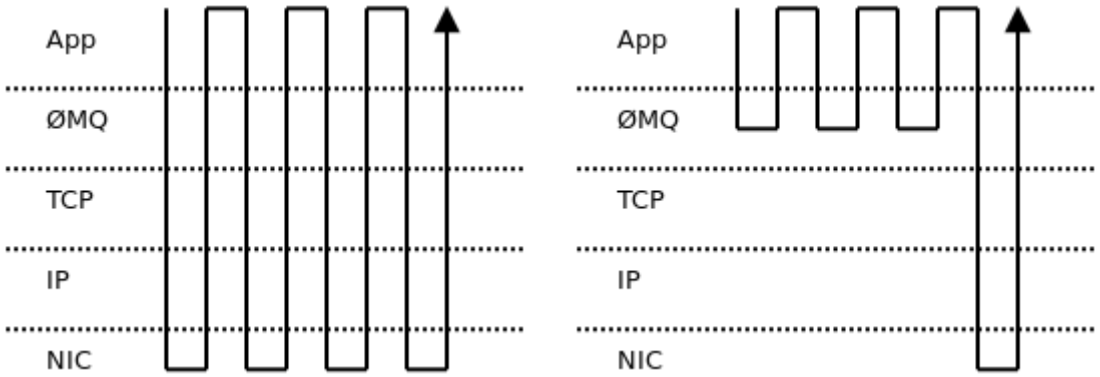
Arquitetura, Mensagem e *Batching*



https://github.com/zeromq/libzmq/blob/master/src/pipe.hpp
https://github.com/zeromq/libzmq/blob/master/src/pipe.cpp
https://github.com/zeromq/libzmq/blob/master/src/session_base.hpp
https://github.com/zeromq/libzmq/blob/master/src/session_base.cpp
https://github.com/zeromq/libzmq/blob/master/src/i_engine.hpp
https://github.com/zeromq/libzmq/blob/master/src/tcp_listener.hpp
https://github.com/zeromq/libzmq/blob/master/src/tcp_listener.cpp



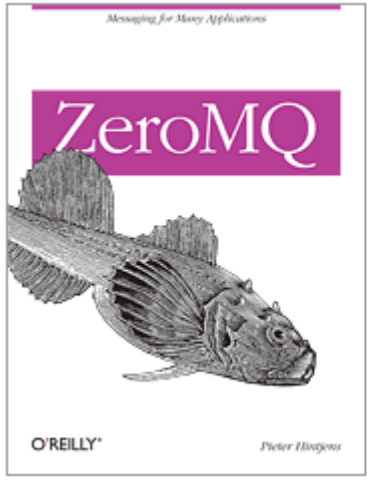
https://github.com/zeromq/libzmq/blob/master/src/msg.hpp
https://github.com/zeromq/libzmq/blob/master/src/msg.cpp



http://aosabook.org/en/zeromq.html

Referências

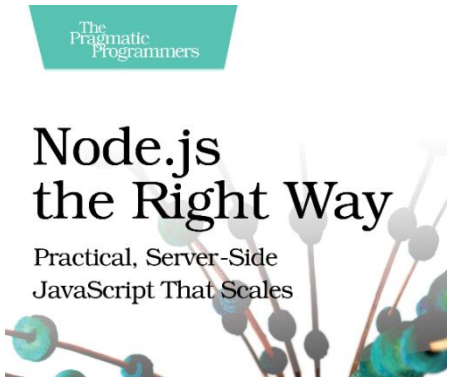
- http://zeromq.org
- ZeroMQ
  - http://shop.oreilly.com/product/0636920026136.do
  - http://zguide.zeromq.org/



ZeroMQ
Messaging for Many Applications
By Pieter Hintjens
Publisher: O'Reilly Media
Final Release Date: March 2013
Pages: 516

5.0
Read 6 Reviews | Write a Review

- Node.js the Right Way
  - https://pragprog.com/book/jwnode/node-js-the-right-way
  - Um capítulo dedicado a ZeroMQ
    - Robust Messaging Services
      - Advantages of ØMQ
      - Importing External Modules with npm
      - Message-Publishing and -Subscribing
      - Responding to Requests
      - Routing and Dealing Messages
      - Clustering Node.js Processes
      - Pushing and Pulling Messages
      - Wrapping Up



Fabio Galuppo, M.Sc.

http://fabiogaluppo.com e http://simplycpp.com/

fabiogaluppo@acm.org

@FabioGaluppo

Microsoft MVP Visual Studio and Development Technologies

https://mvp.microsoft.com/en-us/PublicProfile/9529