Mensagens distribuídas com alto desempenho

ØMQ e aplicações financeiras em larga escala

Fabio Galuppo, M.Sc.

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

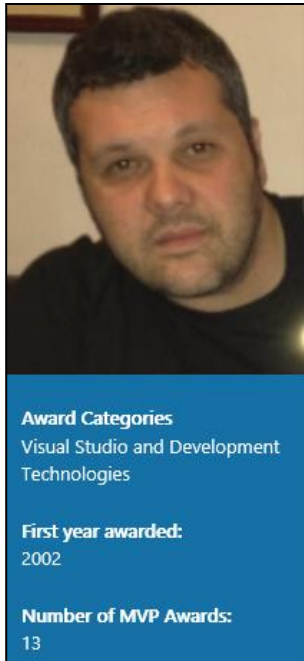
fabiogaluppo@acm.org

@FabioGaluppo

Microsoft MVP Visual Studio and Development Technologies

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

http://bit.ly/zmq_qconsp_2017



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?

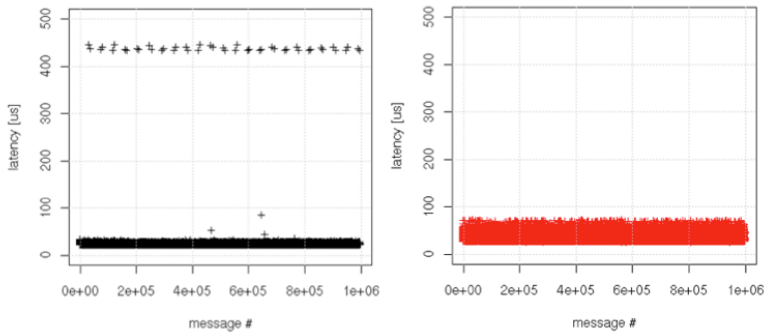
Since ZeroMQ is free software we don't track who uses it. However, some organizations that we know use it are: AT&T, Cisco, EA, Los Alamos Labs, NASA, Weta Digital, Zynga, Spotify, Samsung Electronics, Microsoft, and CERN.

- 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

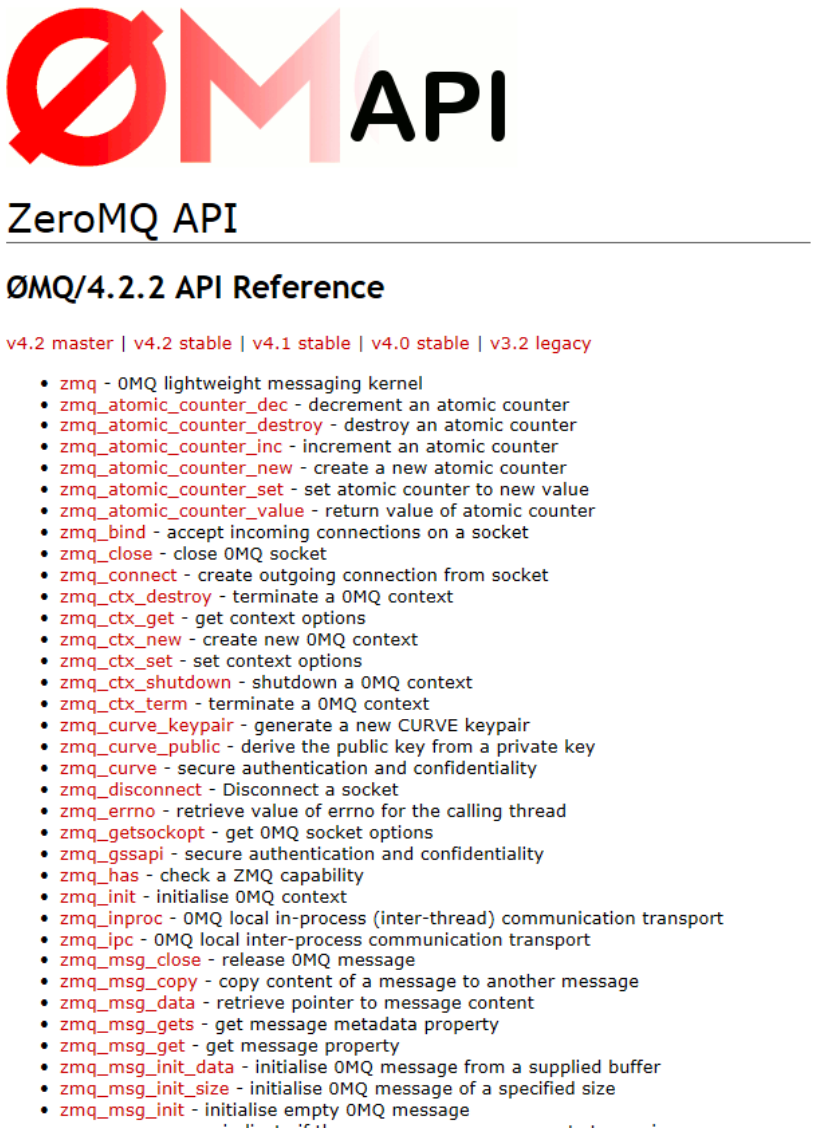
	patterns	QoS	resources	performance	user friendly	community	score
CORBA	✗	✓	✗	✓	✗	✗	2
Ice	✓	✓	✗	✓	✓	✓	5
Thrift	✗	✗	✓	✓	✗	✗	2
ZeroMQ	✓	✓	✓	✓	✓	✓	6
YAMI4	✓	✓	✓	✗	✓	✗	4
RTI	✗	✓	✗	✓	✗	✓	3
Qpid	✗	✓	✗	✗	✓	✓	3

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

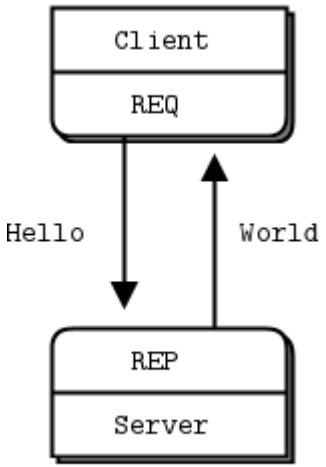
- http://api.zeromq.org/
  - zmq_ctx_new - create new 0MQ context
  - zmq_term - terminate 0MQ context
  - zmq_socket - create 0MQ socket
  - zmq_close - close 0MQ socket
  - zmq_bind - accept incoming connections on a socket
  - zmq_connect - create outgoing connection from socket
  - zmq_send - send 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$
```

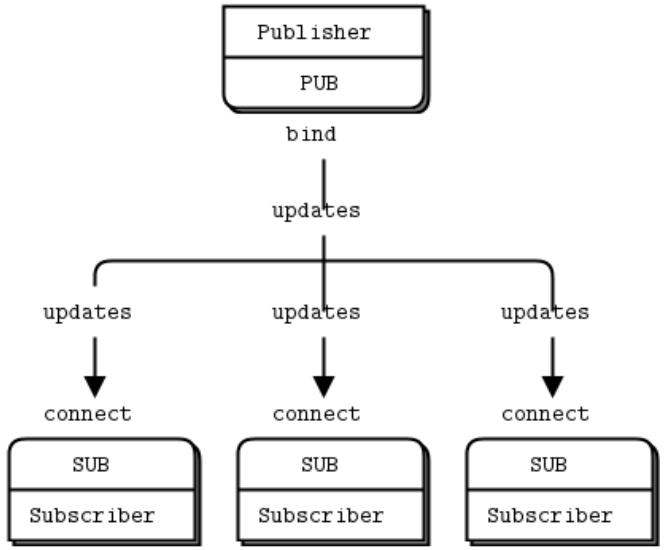
Publisher-Subscriber Pattern

sample-3 — pub.exe — 80×24

```
Fabios-MacBook-Pro:sample-3 fabiogaluppo$ clang++ -std=c++11 -I../include -I./cp
pzmq/include -lzmq pub.cpp -o ./bin/pub.exe
Fabios-MacBook-Pro:sample-3 fabiogaluppo$ ./bin/pub.exe
ZMQ version = 4.2.3
Publishing...
[CTRL + C] to finish...
GOOG 1000
MSFT 1001
APPL 1002
GOOG 1003
MSFT 1004
APPL 1005
APPL 1006
MSFT 1007
```

sample-3 — bash — 80×24

```
Fabios-MacBook-Pro:sample-3 fabiogaluppo$ sudo javac -d bin -cp .:../refs/zmq.ja
r Sub.java
Password:
Fabios-MacBook-Pro:sample-3 fabiogaluppo$ java -Djava.library.path="/Users/fabio
galuppo/zmq/samples/refs" -cp .:./bin:../refs/zmq.jar Sub 5 "GOOG " "MSFT "
ZMQ version = 4.2.3
Listening...
[CTRL + C] to finish...
GOOG 1000
MSFT 1001
GOOG 1003
MSFT 1004
MSFT 1007
Fabios-MacBook-Pro:sample-3 fabiogaluppo$
```



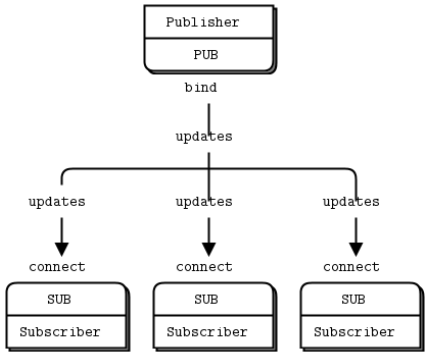
Publisher-Subscriber Pattern

```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)

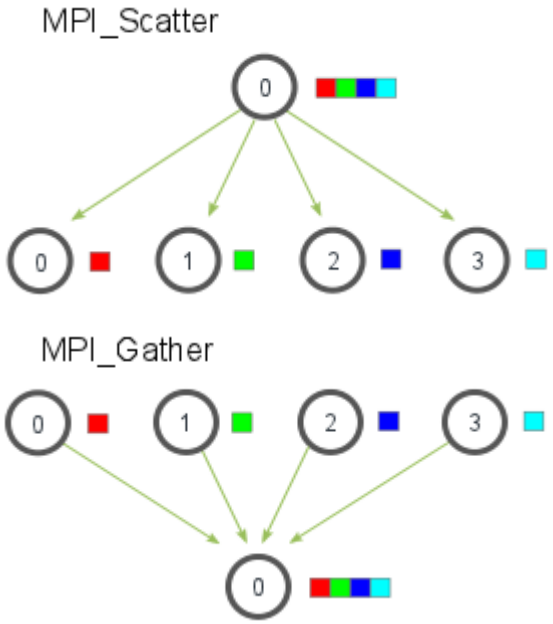
```java
Context zmqContext = ZMQ.context(1);

Socket subscriberSocket = zmqContext.socket(ZMQ.SUB);
//subscriberSocket.connect("tcp://localhost:60001");
subscriberSocket.connect("ipc://pub.ipc");

System.out.println("Listening...");
System.out.println("[CTRL + C] to finish...");

for (int i = 1; i < args.length; ++i) {
    subscriberSocket.subscribe(args[i].getBytes());
}

for (int i = 0; i < N; ++i) {
    byte[] data = subscriberSocket.recv(0);
    String text = new String(data);
    System.out.println(text);
}

subscriberSocket.close();
zmqContext.close();
```

Subscriber – Java Binding (jzmq)

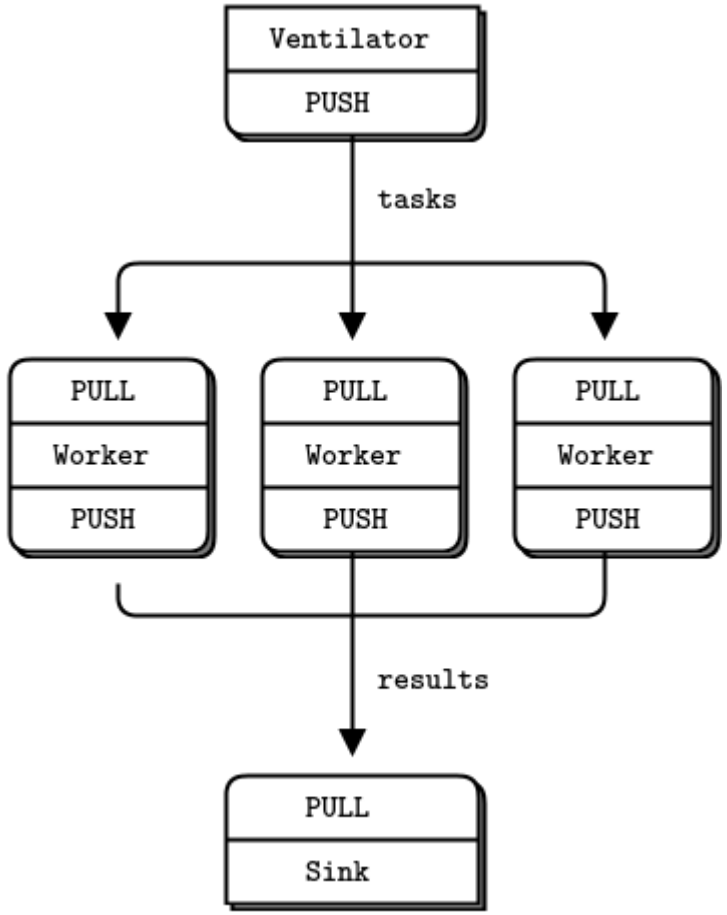


Parallel Pipeline Pattern

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



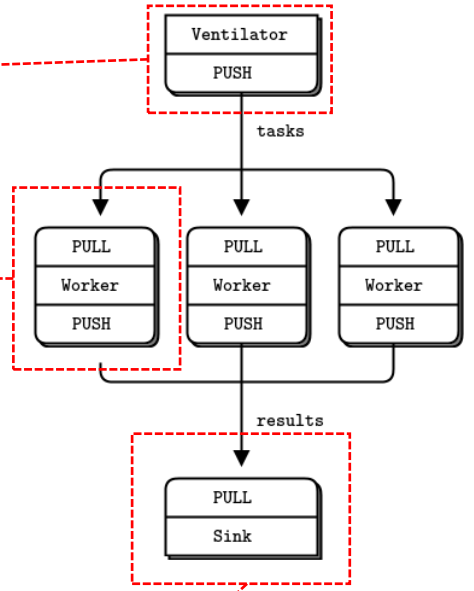
http://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/



Parallel Pipeline Pattern

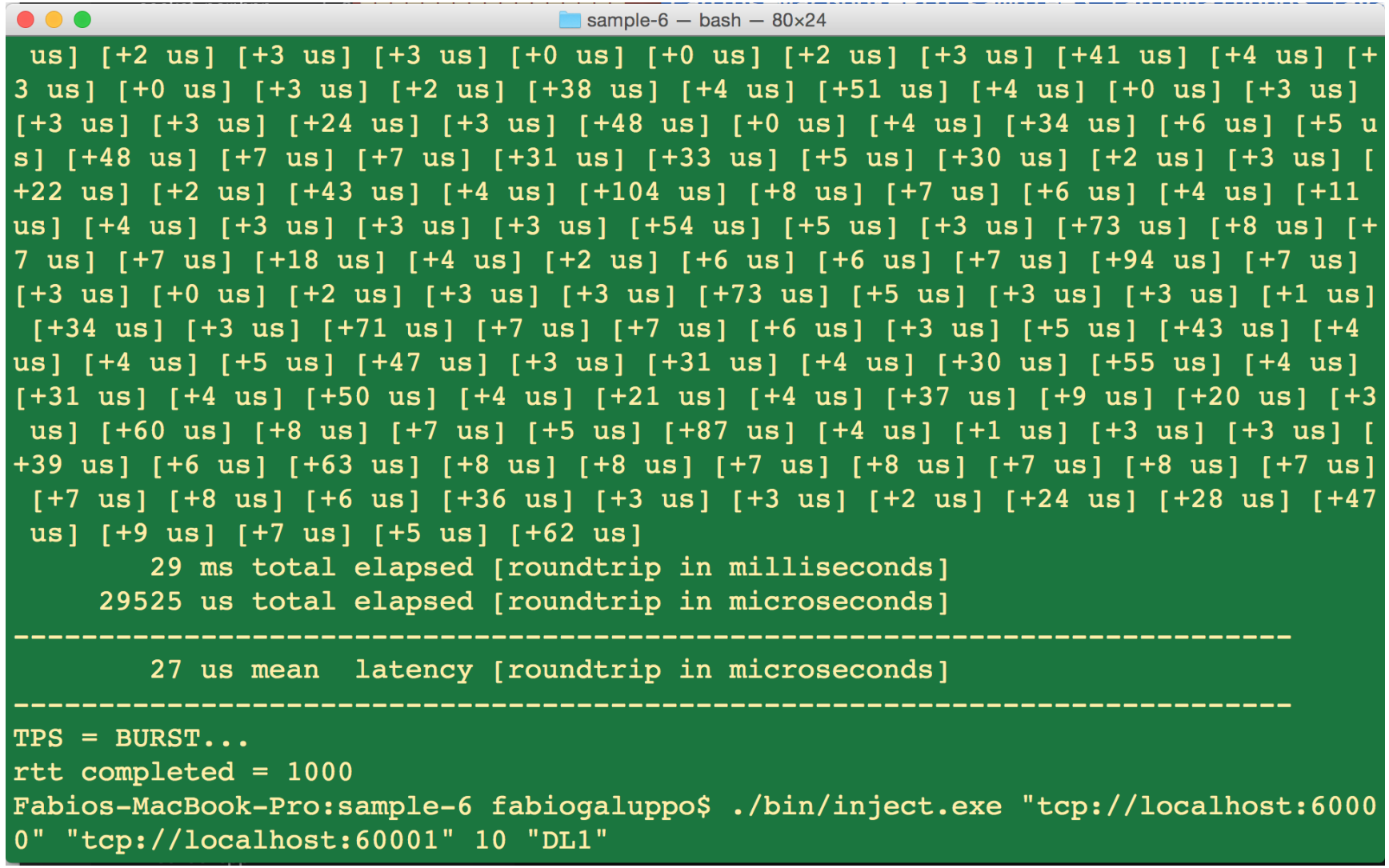
```cpp
zmq::context_t zmq_context(2);
zmq::socket_t source_socket(zmq_context, ZMQ_PUSH);
zmq::socket_t sink_socket(zmq_context, ZMQ_PUSH);

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
}
```

Dealer-Router



https://github.com/fabiogaluppo/samples/blob/master/events/qconsp2017/code/sample-6/inject2.hpp

Dealer-Router

```
sample-5 — java — 80×24
Fabios-MacBook-Pro:sample-5 fabiogaluppo$ sudo javac -d bin -cp .:../refs/zmq.ja
r Recept.java
Fabios-MacBook-Pro:sample-5 fabiogaluppo$ java -Djava.library.path="/Users/fabio
galuppo/zmq/samples/refs" -cp .:./bin:../refs/zmq.jar Recept "tcp://*:60001" "tc
p://*:60000"
ZMQ version = 4.2.3
Listening...
```

Configuring router…

```
Context zmqContext = ZMQ.context(2);

SocketPair router = SocketPair.newRouter(zmqContext, routerSenderBindAddress, routerReceiverBindAdd
    Socket senderSocket = socketPair.getSenderSocket();
    Socket receiverSocket = socketPair.getReceiverSocket();

    senderSocket.setRouterMandatory(true);
    senderSocket.setLinger(0);
    senderSocket.setTCPKeepAlive(-1);
    senderSocket.setSndHWM(1000);
    senderSocket.setRcvHWM(1000);
    senderSocket.setAffinity(1);
    senderSocket.setLongSockoptUnsafe(ZMQ_ROUTER_HANDOVER, 1);

    receiverSocket.setLinger(0);
    receiverSocket.setTCPKeepAlive(-1);
    receiverSocket.setSndHWM(1000);
    receiverSocket.setRcvHWM(1000);
    receiverSocket.setAffinity(2);
    receiverSocket.setLongSockoptUnsafe(ZMQ_ROUTER_HANDOVER, 1);
});

ByteBuffer bbDecoder  = ByteBuffer.allocateDirect(1).order(ByteOrder.nativeOrder());
ByteBuffer bbDealerId = ByteBuffer.allocateDirect(3).order(ByteOrder.nativeOrder());
ByteBuffer bbPayload  = ByteBuffer.allocateDirect(512).order(ByteOrder.nativeOrder());
```

Receiving…

```
SenderRunner senderRunner = new SenderRunner(router);
Thread senderThread = new Thread(senderRunner);
senderThread.start();

int count = 0;
while(!Thread.currentThread().isInterrupted()) {
    poller.poll(0);
    if (poller.pollin(0)) {
        while(!SocketPairOps.tryBulkReceive(router, bbDealerId, bbDecoder, bbPayload));
        senderRunner.sendBack(clone(bbDealerId), clone(bbDecoder), clone(bbPayload));
    }
}
```

Sending back…

```
@Override
public void run() {
    while (!Thread.currentThread().isInterrupted()) {
        if (!Q.isEmpty()) {
            Buffers buffers = Q.poll();
            while(!SocketPairOps.tryBulkSend(router, buffers.bbClientId, buffers.bbDecoder, buffers.bbPayload));
        }
    }
}
```

https://github.com/fabiogaluppo/samples/blob/master/events/qconsp2017/code/sample-5/Recept.java

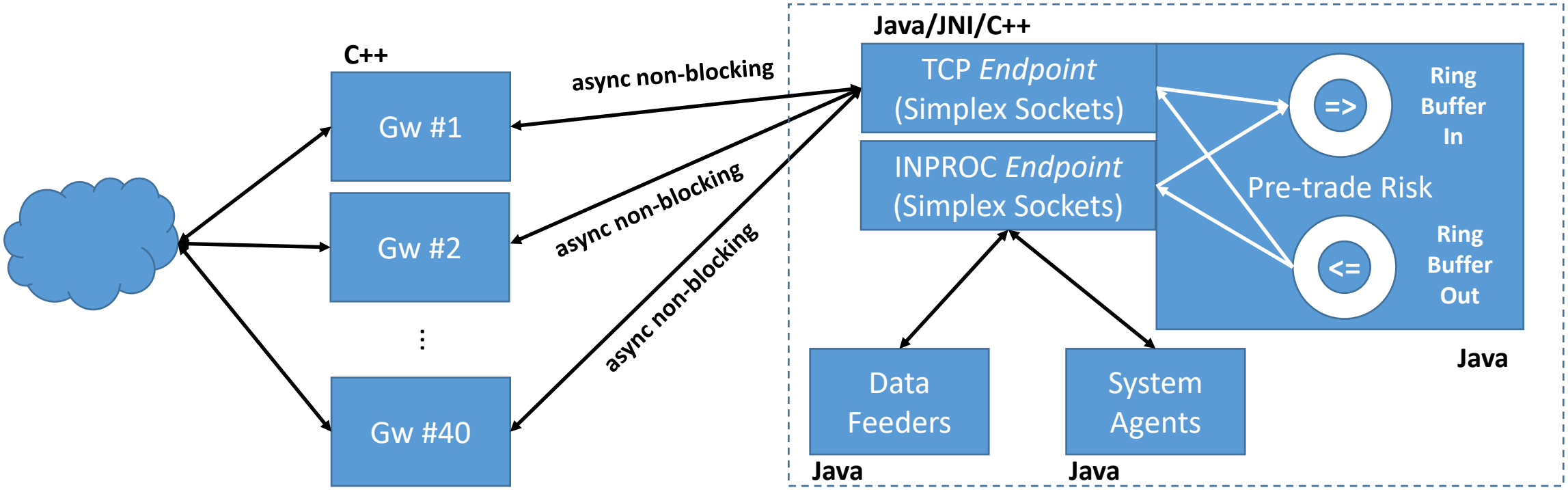
Contexto de Aplicação

- Sistema de Risco Pré-Negociação BM&FBovespa
  - Permite que seja estabelecido limites operacionais para ofertas (compra ou venda) enviadas à Bolsa
  - Mensagens são transmitidas e interceptadas pelos Gateways de Entrada de Ordens que repassam para o Sistema de Avaliação de Risco (Pre-trade Risk)
  - Uma camada de serviço é responsável por fornecer os *endpoints* de entrada e saída destas mensagens.
    - Comunicação assíncrona e não bloqueante
    - Os principais *endpoints* trocam dados via TCP, IPC ou INPROC em formatos binários:

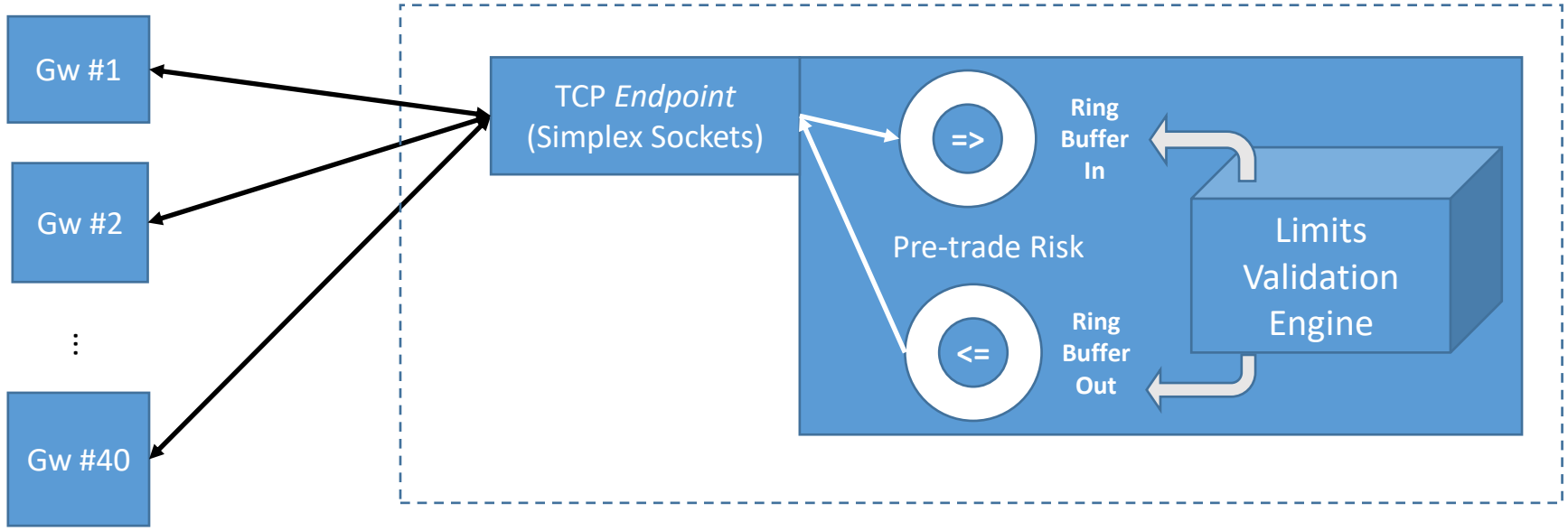
Client Id (3 Bytes)	Encoder/Decoder Id (1 Byte)	Payload (BLOB)

    - Roteamento para os *engines* de validação de limites e regras de negócio

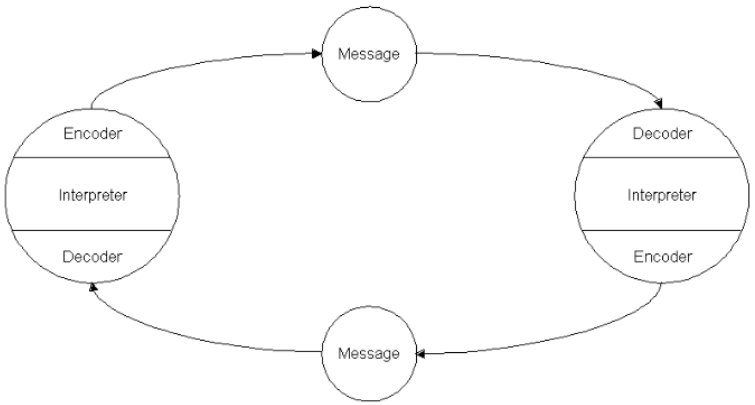
Arquitetura para Distribuição de Mensagens



Modelo de Comunicação



Schramm's Model of Communication, 1954



https://www.businesstopia.net/communication/schramms-model-communication

- Similar ao modelo de Schramms, porém adaptado para "N *senders*"
  - *Inbound Ring Buffer* determina a sequência de processamento
  - *Engine*
    - Retira do *Inbound Ring Buffer*
    - Delega para processador
    - Insere a resposta no *Outbound Ring Buffer*

Event Loop

In computer science, the **event loop**, message dispatcher, message **loop**, message pump, or run **loop** is a programming construct that waits for and dispatches **events** or messages in a program.

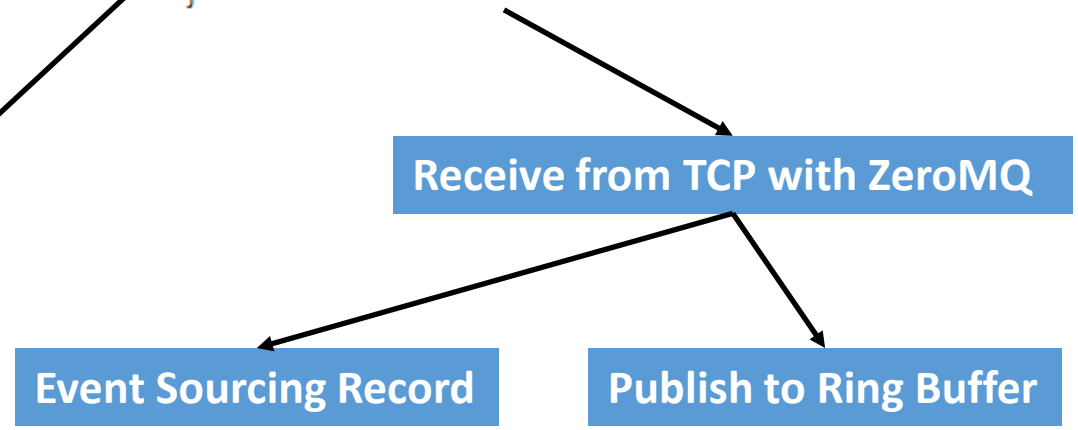
Event loop - Wikipedia
https://en.wikipedia.org/wiki/Event_loop

- 2 Threads de Event Loop
  - 1 para receber mensagens e 1 para enviar mensagens

```java
@Override
public final void run() {
    LOGGER.info("Entering TransportFacade event loop");

    // start event sourcing
    if (isJournalingEnabled()) {
        inboundQueueManager.start();
        outboundQueueManager.start();
    }

    // other thread (send messages)
    senderThread.start();

    // this thread message loop (receive messages)
    try {
        while (isTransportFacadeReady.get()) {
            receiveAndProcessAdminMessages();
            receiveAndProcessBusinessMessages();
        }
    } catch (Exception e) {
        LOGGER.fatal("Receiver Unexpected Exception", e);
    }

    closeResources();
    closeResourcesLatch.countDown();
}
```

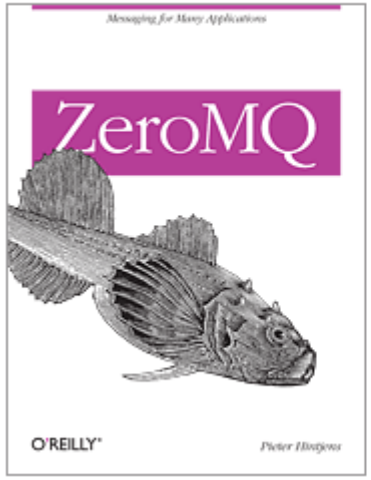
```java
receiverThread = affinityThreadFactory.newThread(this);
senderThread = affinityThreadFactory.newThread(new SendRunner());
```

```java
receiverPoller.poll(0);
if (tcpRouterBusinessListenerAndPublisher()) {
    handled = true;
}
```



Referências

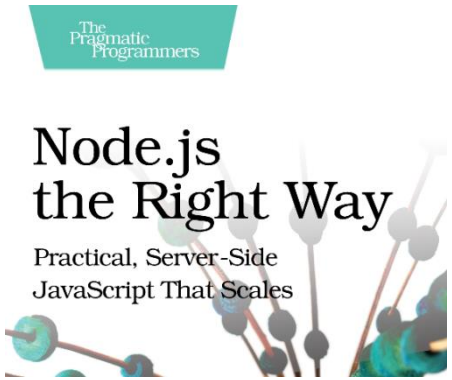
- http://zeromq.org
- ZeroMQ
  - http://shop.oreilly.com/product/0636920026136.do
  - http://zguide.zeromq.org/



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



Mensagens distribuídas com alto desempenho

ØMQ e aplicações financeiras em larga escala

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

http://bit.ly/zmq_qconsp_2017