ZeroMQ

Эффективный framework для разработки распределенных приложений

Зачем нам нужен ZeroMQ ?

- Есть три основных подхода к разработке распределенных приложений :
  - Большие бизнес приложения обычно используют брокер очередей или SOA (Service Oriented Architecture)
  - Небольшие проекты используют низкоуровневый сетевой обмен

- Все три подхода имеют недостатки :
- Брокер очередей – еще один Big Box требующий поддержки, администрирования, выделенного аппаратного решения
- SOA – имеет проблемы с производительностью и отказоустойчивостью
- Написать качественное решение на уровне сокетов крайне сложно

Что предлагает ZeroMQ

- Простой асинхронный framework для обмена сообщениями выше уровнем чем классические сокеты, но ниже чем классические messaging API
- Высокая производительность (существенно выше чем у классических messaging брокеров)
- Реализация большого количества протоколов
- Возможность самостоятельно разрабатывать легковесные прокси-брокеры
- Возможность реализации большого количества паттернов обмена
- ZeroMQ написан на C и предлагает bindings на большое количество языков программирования

На что это похоже

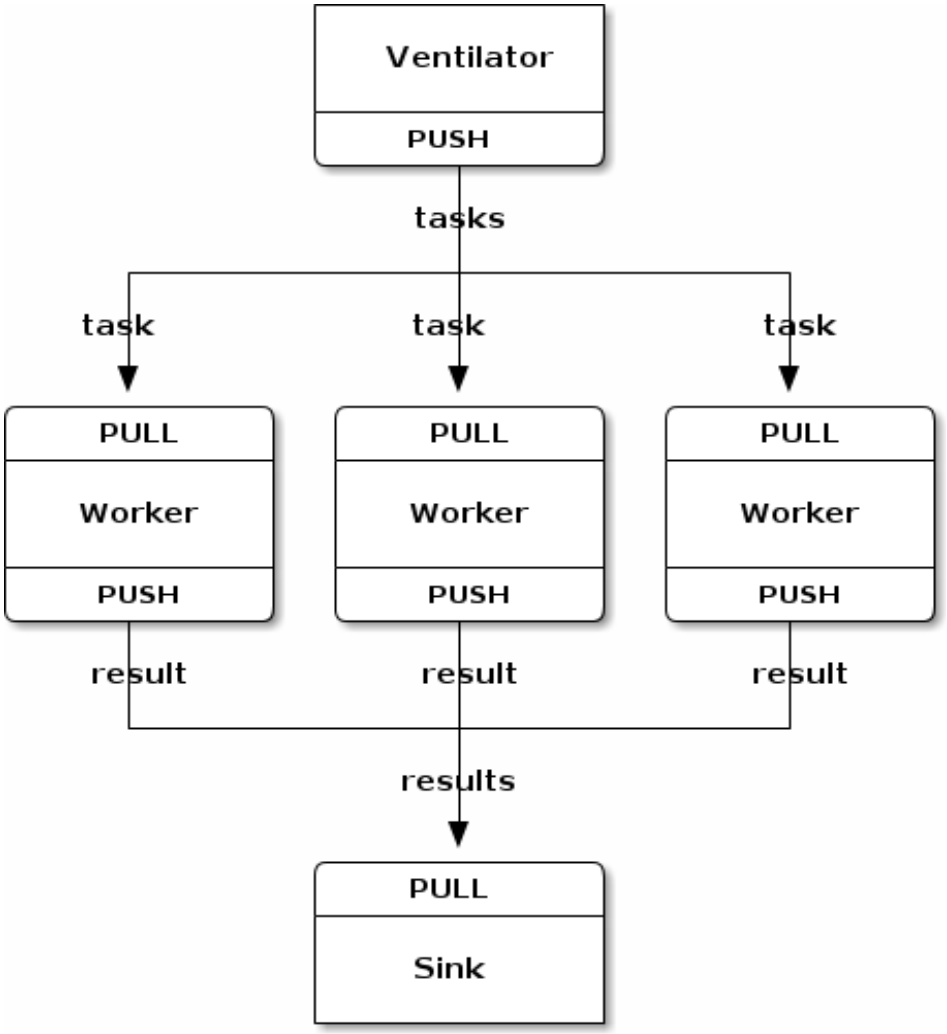


Figure 5 – Parallel Pipeline

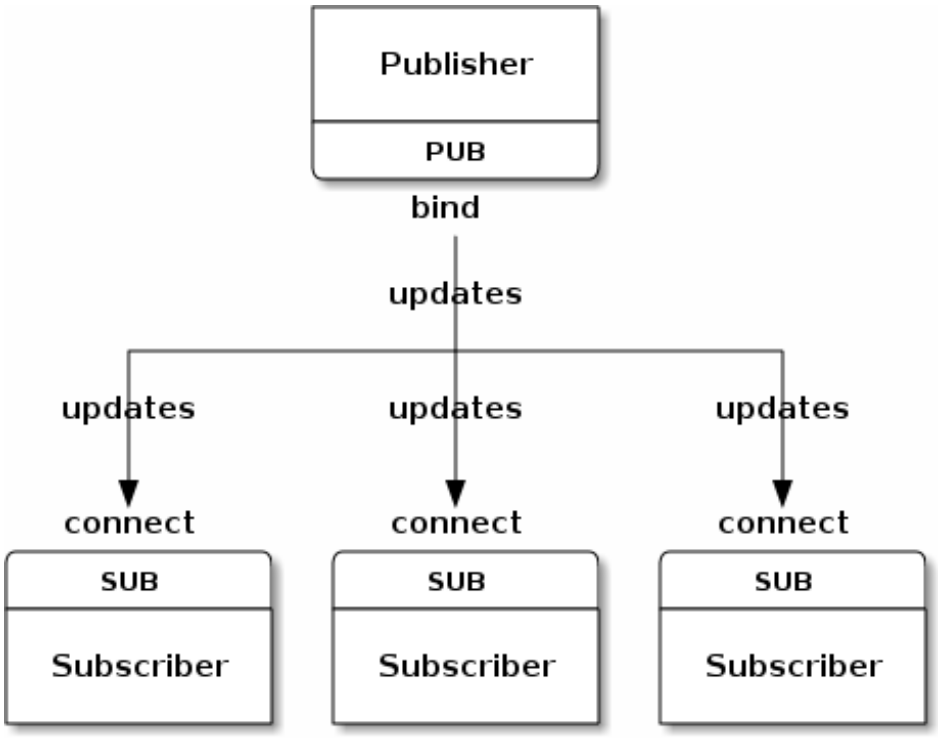
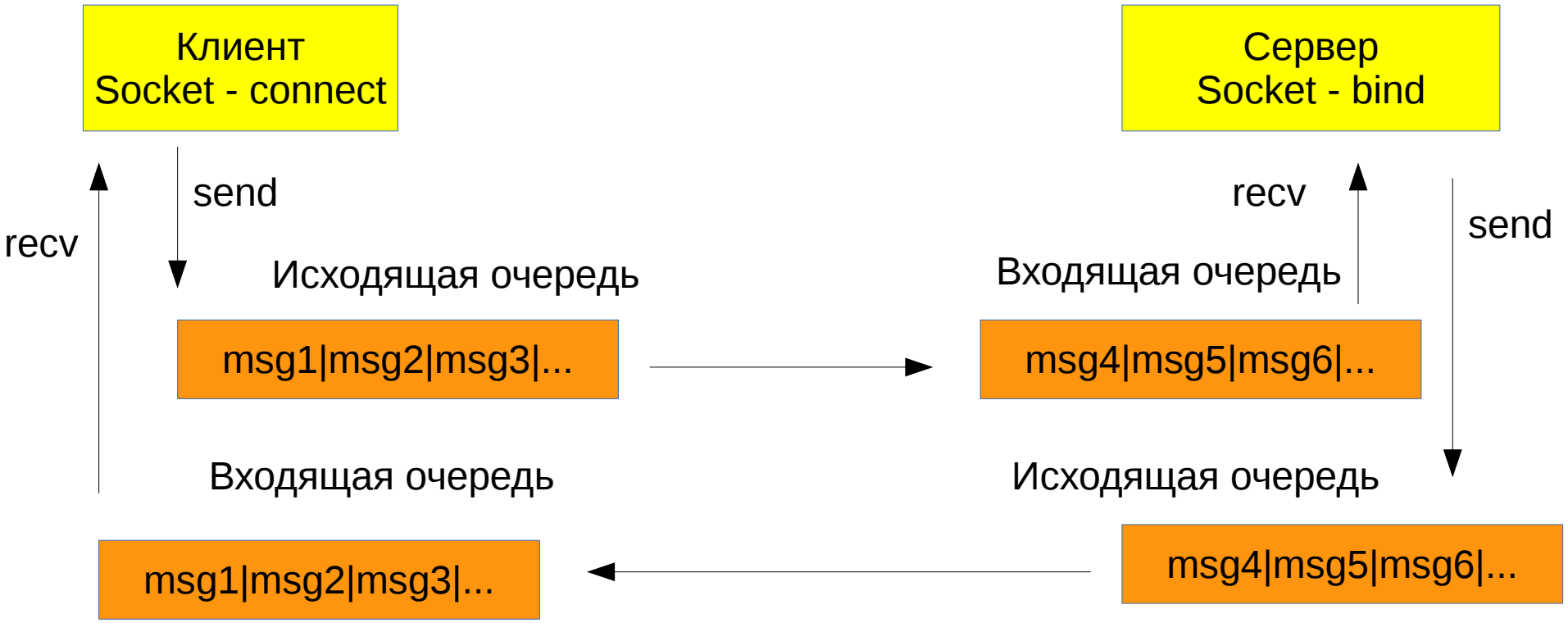


Figure 4 – Publish-Subscribe

Базовые термины ZeroMQ

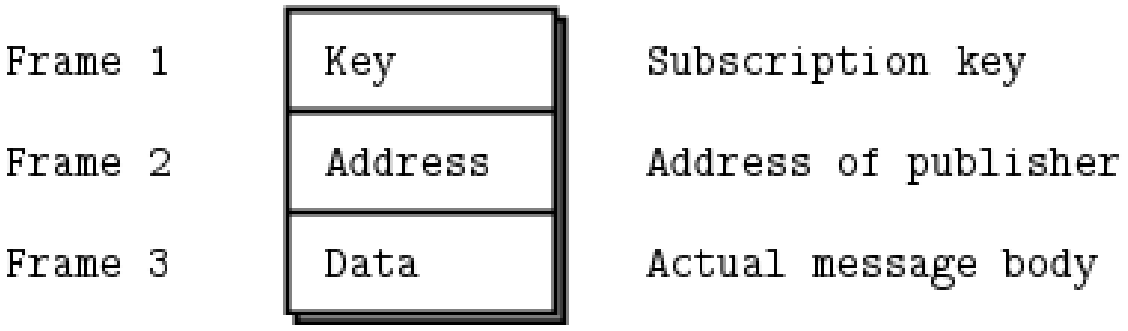
- Context – базовый класс framework ZeroMQ.
  - Создает и удаляет сокеты
  - Управляет настройками ZeroMQ
- Socket – точка подключения одной системы к другой.
- Каждая сторона обмена создает свой сокет.
- Одна сторона создает статическую точку подключения – делает bind
- Вторая подключается к ней с помощью метода connect.
- В подключенный сокет можно послать сообщение а также получить сообщение (задав таймаут)
- Сообщения кладутся в очередь на отправляющей стороне. Асинохронно передаются на принимающую и кладутся в очередь там
- Сокеты имеют типы. Каждый тип реализует определенный паттерн обмена данными

Принципиальная схема



Сообщение ZeroMQ

- Сообщение состоит из фреймов.
- Каждый фрейм содержит массив байтов
- Сообщение передается атомарно, т.е. Либо доходят все фреймы, либо ни одного
- Строки передаются как length based
- Пример сообщения



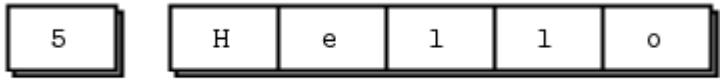


Схема типичного сервера ZeroMQ

- Создаем ZContext или ZMQ.Context
- С помощью методов Context.socket(<тип сокета>) или ZContext.createSocket(<тип сокета>) создаем сокет
- В бесконечном цикле читаем из сокета данные и отвечаем или посылаем данные в другие сокеты
- Посылать можно набор байтов, строку и сообщение Zmsg(набор фреймов)
- Массив и строка посылаются методами сокета sendXXX, принимаются методами сокетв recvXXX
- ZMsg отправляется и посылается своими методами ZMsg.send и ZMsg.recvMsg

Чтение из нескольких сокетов

- Один сокет ZeroMQ должен использоваться только в одном потоке.
- В случае паттерна, который требует в одном потоке работать с несколькими сокетами используется класс Poller
- Алгоритм работы с Poller :
  - *Poller items = ctx.createPoller(2);*
  - регистрируем сокет
  - *items.register(frontend, Poller.POLLIN);*
  - регистрируем сокет
  - *items.register(backend, Poller.POLLIN);*
  - *читаем событие*
  - *items.poll();*
  - Узнаем на какой сокет пришло сообщение
  - if (items.pollin(**0**)) {…} //сообщение пришло на первый сокет

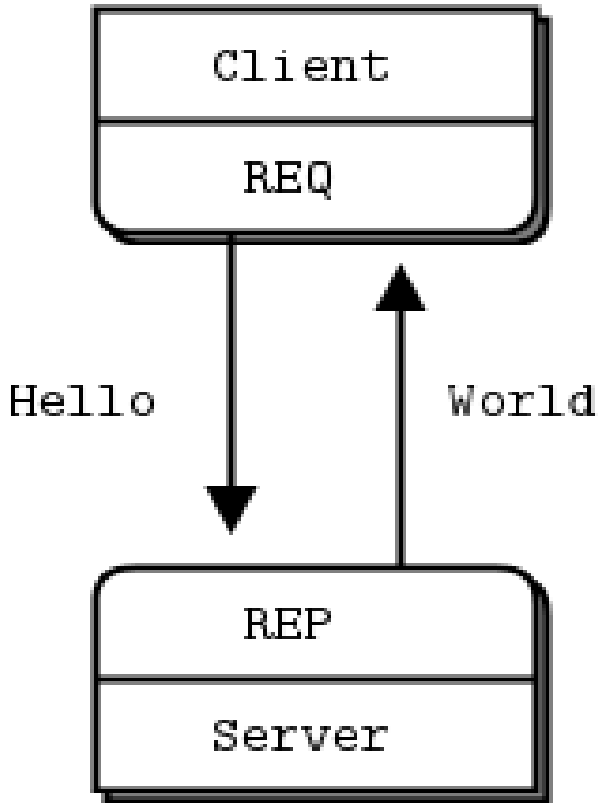
Виды сокетов. REQ

- Сокеты REQ служат для отправки сообщений и для приема ответов
- После отправки сообщения следует запросить ответ. Повторная отправка сообщения не разрешена
- В начало сообщения добавляется пустой фрейм

Виды сокетов. REP

- REP служат для приема запросов и отправки ответов на них
- В сокет REP можно послать только сообщение ответ и далее ожидать следующего сообщения
- В момент приема сообщения ожидает что первый фрейм будет пустым и удаляет его

Взаимодействие REQ-REP



Взаимодействие запрос-ответ Сервер

```
public class ServerRep {
    public static void main(String[] args) {
        ZContext context = new ZContext();
        ZMQ.Socket socket = context.createSocket(SocketType.REP);
        try {
            socket.bind("tcp://localhost:5555");
            System.out.println("bind!");
            while (!Thread.currentThread().isInterrupted()) {
                String req = socket.recvStr();
                socket.send("reply!" + req);
            }
        } finally {
            context.destroySocket(socket);
            context.destroy();
        }
    }
}
```

Взаимодействие запрос-ответ Клиент.

```
public class ClientReq {
    public static void main(String[] args) {
        ZContext context = new ZContext();
        ZMQ.Socket socket=null;
        try {
            System.out.println("connect");
            socket = context.createSocket(SocketType.REQ);
            socket.connect("tcp://localhost:5555");
            for (int i = 0; i < 10; i++) {
                socket.send("request" + i, 0);
                String reply = socket.recvStr();
                System.out.println("reply "+i+" result="+reply);
            }
        } finally {
            context.destroySocket(socket);
            context.destroy();
        }
    }
}
```

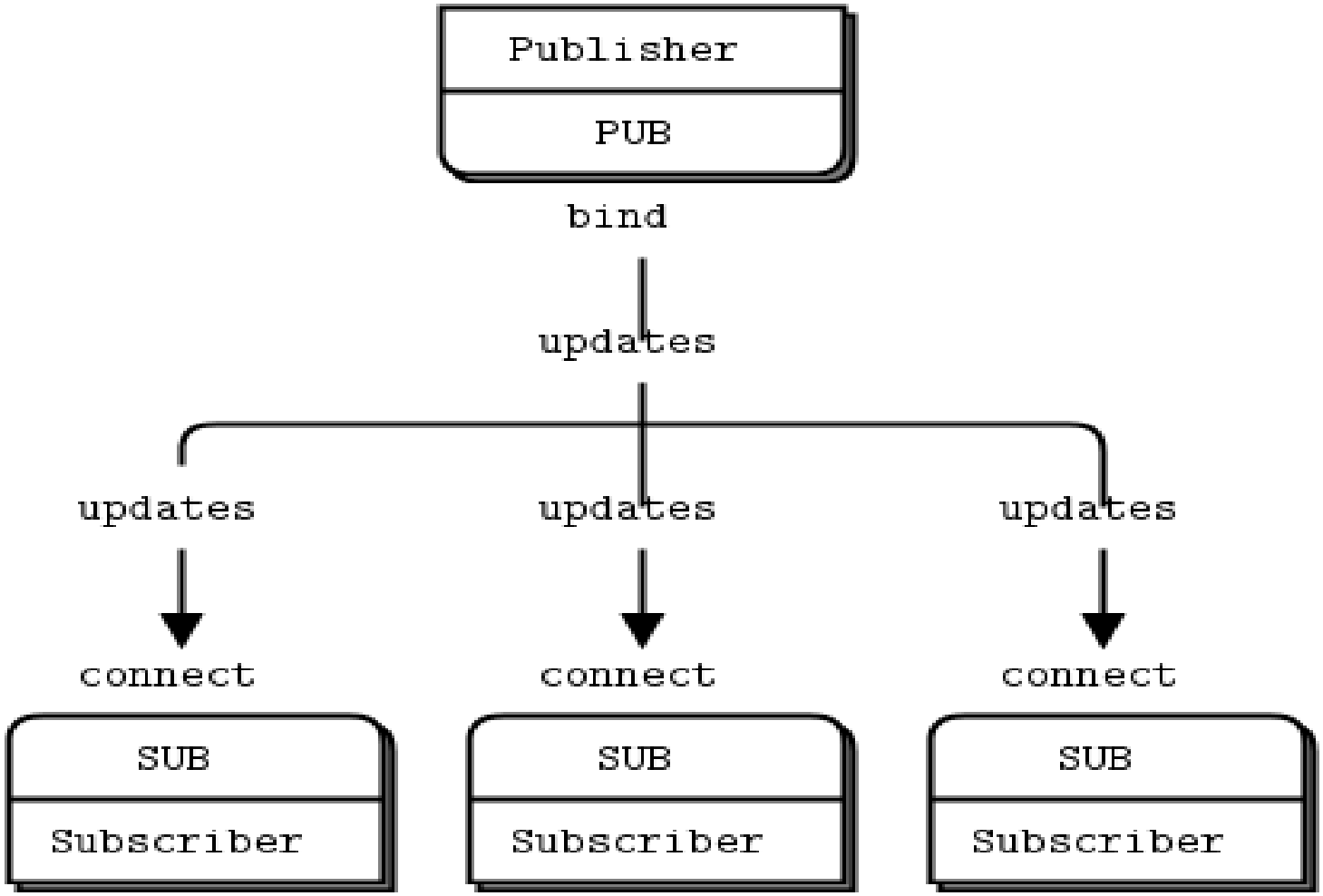
Виды сокетов.PUB

- PUB предназначен для взаимодействия publish-subscriber в ходе которого publisher рассылает сообщения которые принимаются subscriber-ам
- В PUB можно только отправлять сообщения.
- Нет способа узнать, приняли ли наше сообщения
- Неизвестно количество subscriber-ов

Виды сокетов.SUB

- С помощью сокетов SUB мы можем подписаться на получение событий от publisher
- Следует учесть что момент установки соединения и получения первых сообщений не гарантируется.
- С помощью метода subscribe мы указываем фильтр на получаемые сообщения :
  - Все сообщения которые будут нами получены будут начинаться на строку указанную в методе subscribe

Взаимодействие PUB-SUB



Клиент SUB

```
  ZMQ.Context context = ZMQ.context(1);
  // Socket to talk to server
  System.out.println("Collecting updates from
   weather server");
 ZMQ.Socket subscriber =
context.socket(SocketType.SUB);
  subscriber.connect("tcp://localhost:5556");


// Subscribe to zipcode, default is NYC, 10001
  String filter = (args.length > 0) ? args[0] :
"10001 ";
  subscriber.subscribe(filter.getBytes());
// Process 100 updates
int update_nbr;
long total_temp = 0;
```

```
for (update_nbr = 0; update_nbr < 100; update_nbr+
+) {
    String string = subscriber.recvStr(0).trim();
    StringTokenizer sscanf = new
StringTokenizer(string, " ");
    int zipcode = Integer.valueOf(sscanf.nextToken());
    int temperature =
Integer.valueOf(sscanf.nextToken());
    int relhumidity =
Integer.valueOf(sscanf.nextToken());
   total_temp += temperature;
}
System.out.println("Average temperature for zipcode
'"
        + filter + "' was " + (int) (total_temp /
update_nbr));
subscriber.close();
context.term();
```

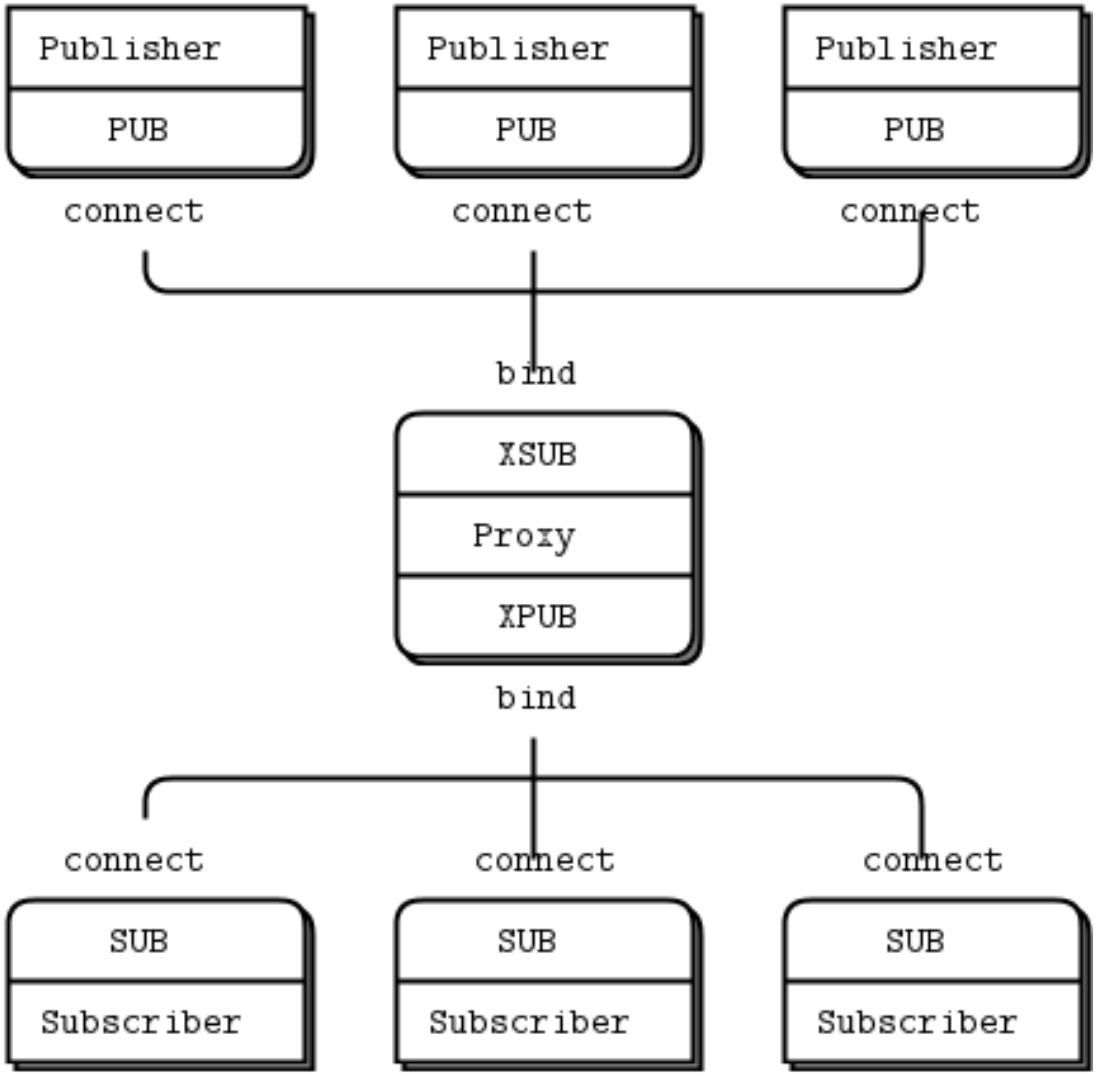
Сервер PUB

```
ZMQ.Context context = ZMQ.context(1);
ZMQ.Socket publisher = context.socket(SocketType.PUB);
publisher.bind("tcp://*:5556");
publisher.bind("ipc://weather");
// Initialize random number generator
Random srandom = new Random(System.currentTimeMillis());
while (!Thread.currentThread ().isInterrupted ()) {
    // Get values that will fool the boss
   int zipcode, temperature, relhumidity;
   zipcode = 10000 + srandom.nextInt(10000) ;
   temperature = srandom.nextInt(215) - 80 + 1;
   relhumidity = srandom.nextInt(50) + 10 + 1;
   // Send message to all subscribers
    String update = String.format("%05d %d %d", zipcode, temperature, relhumidity);
   publisher.send(update, 0);
}
publisher.close ();
context.term ();
```

Взаимодействие XPUB-XSUB

- Центральный сервер в случае многочисленных издателей и подписчиков может быть перегружен
- В этом случае можно применить сокеты XPUB-XSUB которые пересылают только подписку.
- Данные между клиентами идут напрямую

XPUB-XSUB


Publisher
PUB
Publisher
PUB
Publisher
PUB
connect
connect
connect
bind
XSUB
Proxy
XPUB
bind
connect
connect
connect
SUB
Subscriber
SUB
Subscriber
SUB
Subscriber

Простой пример прокси

```
//  Prepare our context and sockets
ZMQ.Context context = ZMQ.context(1);
//  Socket facing clients
ZMQ.Socket frontend = context.socket(SocketType.XSUB);
frontend.bind("tcp://*:5559");
//  Socket facing services
ZMQ.Socket backend = context.socket(SocketType.XPUB);
backend.bind("tcp://*:5560");
//  Start the proxy
ZMQ.proxy (frontend, backend, null);
//  We never get here but clean up anyhow
frontend.close();
backend.close();
context.term();
```

Виды сокетов. PUSH.PULL.

- PUSH и PULL сокеты предназначены для односторонней передачи сообщений
  - В PUSH сокет мы отправляем сообщения
  - Из PULL сокета получаем данные
- В случае параллельного подключения нескольких PUSH сокетов к одному PULL или наоборот – сообщения передаются в порядке очереди.

Взаимодействие PUSH-PULL