ZeroMQ

Edit New Page

Jump to bottom

Taeyoung, Kim edited this page on 15 Mar 2017 · 20 revisions

특징

- 임베디드 네트워킹 라이브러리
  (ActiveMQ or RabbitMQ 같은 메시징 브로커가 존재하지 않음)
- 사용자에 익숙한 Socket Style API로 디자인된 라이브러리
- 메시지 수신이 가능하지 않을 때는 Queuing 될 수 있다
  (over-full 큐에 대한 정책은 메시징 패턴에 따라 결정)
- 많은 노력 없이도 복잡한 통신 시스템을 설계할 수 있도록 해주는 메시징 라이브러리
- 분산 또는 동시성 애플리케이션에서 유용하다
- 라우팅이 가능한 메일박스다
- lock-free (lock, semaphores 대기가 필요하지 않음)
- 자동으로 재 연결 시도 (어떤 순서로도 구성요소를 시작 할 수 있음)
  언제든지 네트워크에 참여하고 떠날 수 있는 서비스 지향 아키텍처(SOA)를 만들수 있다.

분산 애플리케이션(Distributed Application)은 거대한 하나의 애플리케이션(Monolithic Application)보다 변화에 대응하기 쉽다.

> server 가 bind() 하지 않은 상태에서 client 가 connect() 하고 메시지를 송신하더라도 queuing 후에 처리된다.
> (메시지 전송 이후에 close() 하더라도 나중에 서버가 bind() 이후에 데이터가 전달됨,
> close() 이후에 데이터 전송을 원하지 않으면 `socket.setsockopt(zmq.LINGER, 0)` 설정)
>
> (connect)(3)PUSH-PULL(1)(bind)
> PUSH-1 에서 3번 전송, PUSH-2 에서 3번 전송, 서버 bind() 순으로 진행 할 때 메시지는 PUSH-1, PUSH-2 한번씩 번갈아 처리한다.
>
> (connect)(1)PUSH-PULL(3)(bind)
> 3개의 PULL 에서 동시에 recv() 대기중인 상태에서 PUSH 에서 전송하게 되면 3중 한개의 PULL만 메시지를 수신한다.

노드간 연결 방식

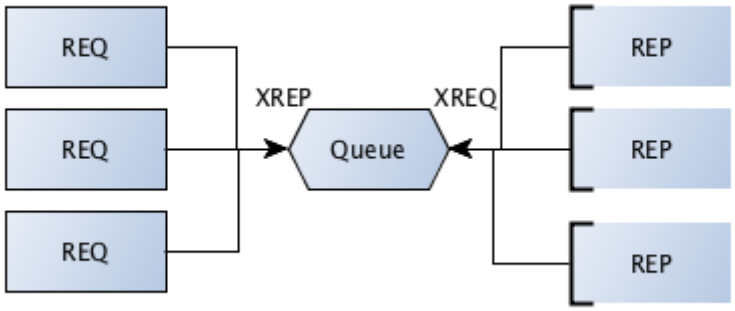
- 한 프로세스의 두 쓰레드
- 한 시스템의 두 프로세스
- 네트워크상의 두 시스템

Messaging Pattern

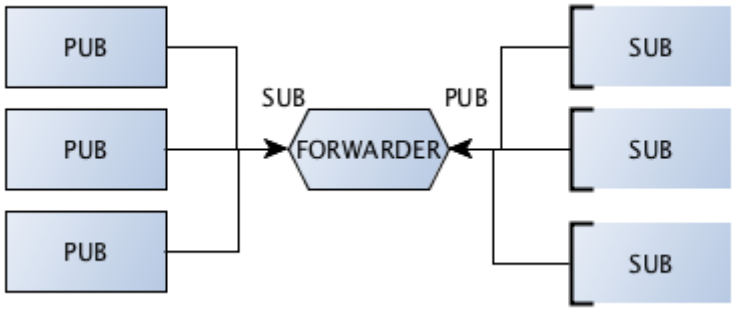
- Request-Reply Pattern
- Publish/Subscribe pattern
- Pipeline Pattern

Device

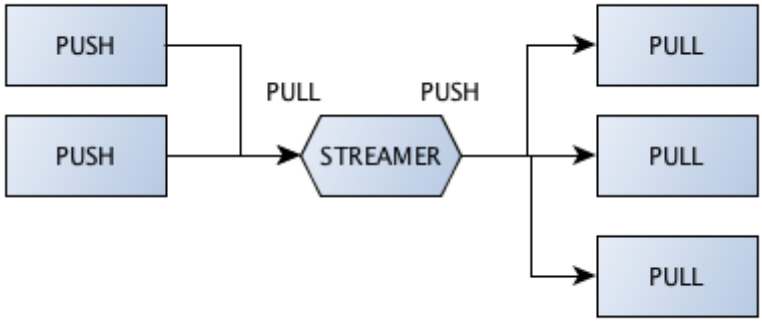
Queue



Forwarder



Streamer



Installation in python

```
# pip install pyzmq
```

버전 확인

```
import zmq

print(zmq.pyzmq_version())
```

ZeroMQ Context

zmq 라이브러리 기능을 사용하기 전에 먼저 생성 되야 한다.

```
import zmq

context = zmq.Context()
```

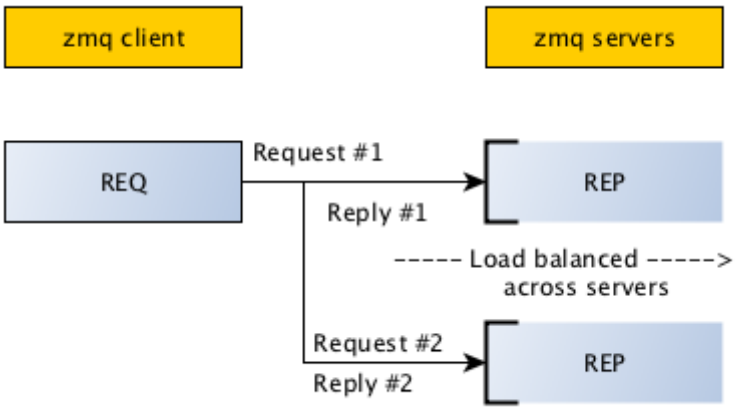
ZeroMQ Sockets

zmq 소켓은 context 를 통해 생성할 수 있다.

```
socket = context.socket(zmq.REP)
```

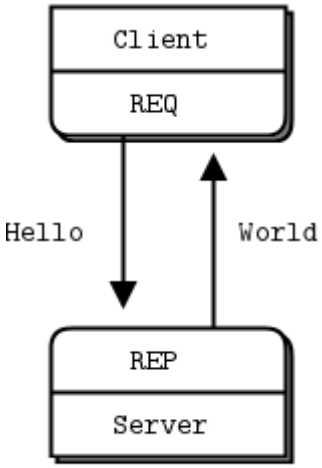
Example

REQ/REP



특징

- REQ socket 은 많은 서버에 연결(connect) 할 수 있다.
- REQ 의 send() 는 응답이 올 때 까지 block 된다.
- REP 의 recv() 는 요청이 수신 될 때 까지 block 된다.



Replay

메시지 응답

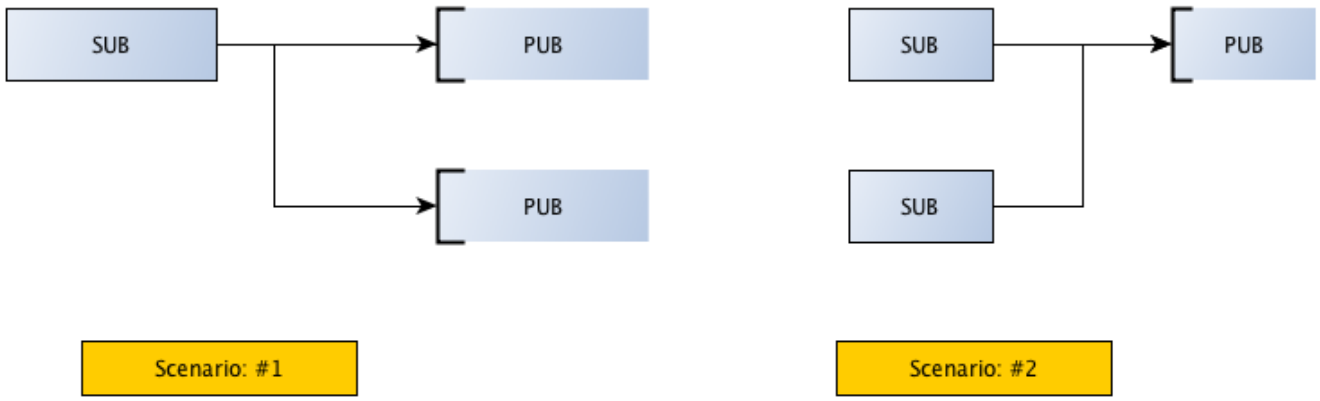
```
import zmq

context = zmq.Context()
socket = context.socket(zmq.REP)
socket.bind('tcp://127.0.0.1:10101')

while True:
    print('recv : ' + socket.recv_string())
    socket.send_string('world')
```

Request

```
import zmq
import time

context = zmq.Context()
socket = context.socket(zmq.REQ)
socket.connect('tcp://127.0.0.1:10101')

while True:
    socket.send_string('hello')
    print('recv : ' + socket.recv_string())
    time.sleep(3)
```

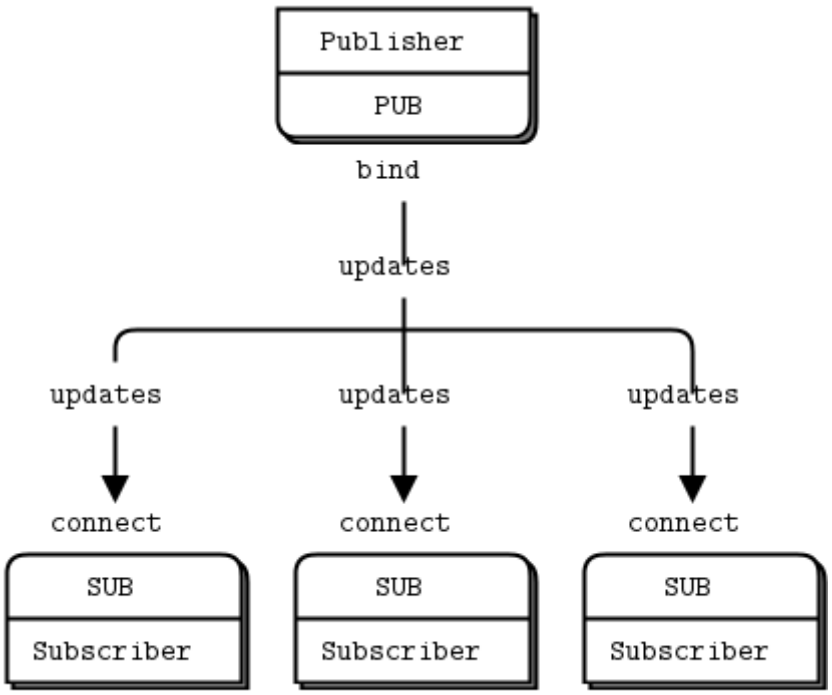
===

Pub/Sub



› 특징

- PUB socket의 경우 연결된 SUB가 없는 경우 메시지는 버려진다.



SUB 의 경우 반드시 setsockopt() 를 사용하여 subscription 을 설정해야 한다.
SUB/PUB 어느곳에서 연결(connect)하든, 바인드(bind) 하든 문제는 되지 않으나, 만약 SUB 소켓을 먼저 바인드(bind) 하고 나중에 PUB 소켓을 연결(connect)하면 SUB 소켓은 오래된 메시지를 받을 수 없게 된다.
그러므로 가능하면 PUB은 바인드(bind), SUB 은 연결(connect) 하는것이 가장 좋다.

Publisher 가 바인딩 후 즉시 메시지를 전송하면, Subscriber 는 데이터를 수신 못할 가능성이 있습니다.
이를 위해서 Subscriber 가 연결하고 준비되기까지 데이터를 발송하지 않도록 동기화 하는 방법을 제공한다.

ØMQ의 PUB-SUB Pattern 특징

- 하나의 Subscriber 는 한 개 이상의 Publisher 에 연결할 수 있다.
- Subscriber 가 없다면 모든 Publisher 의 메시지는 유실 된다.

- Subscriber 에서만 메시지 필터링이 가능하다.

Publisher

3 초에 한번 메시지 전송

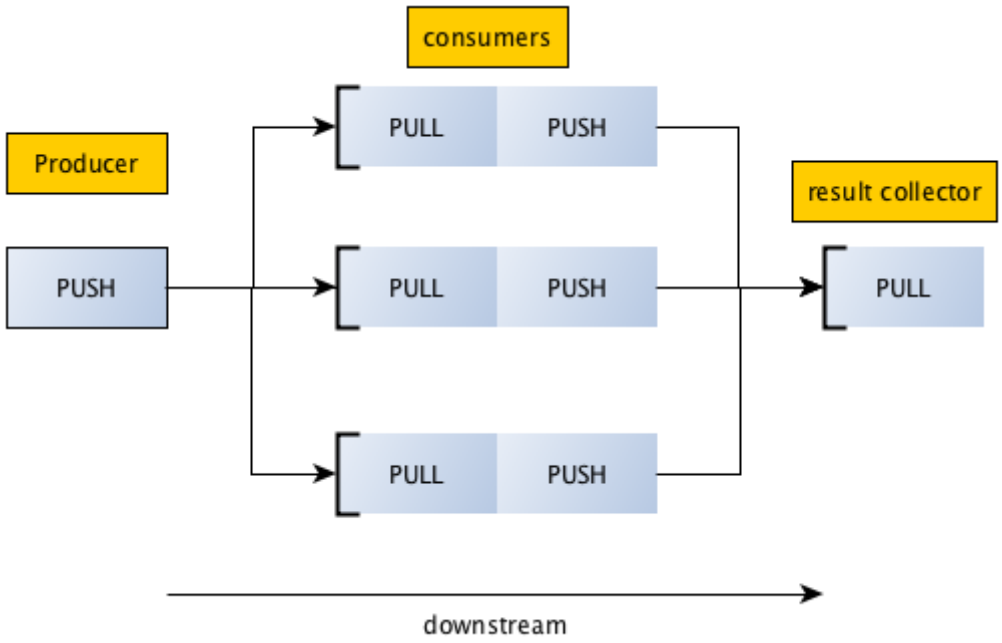
```
import zmq
import time

context = zmq.Context()
socket = context.socket(zmq.PUB)
socket.bind('tcp://127.0.0.1:10100')

while True:
    socket.send_string('Hello')
    time.sleep(3)
```

Subscriber

```
import zmq

context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect('tcp://127.0.0.1:10100')
socket.setsockopt_string(zmq.SUBSCRIBE, '') # it will capture all messages

while True:
    print(socket.recv_string())
```

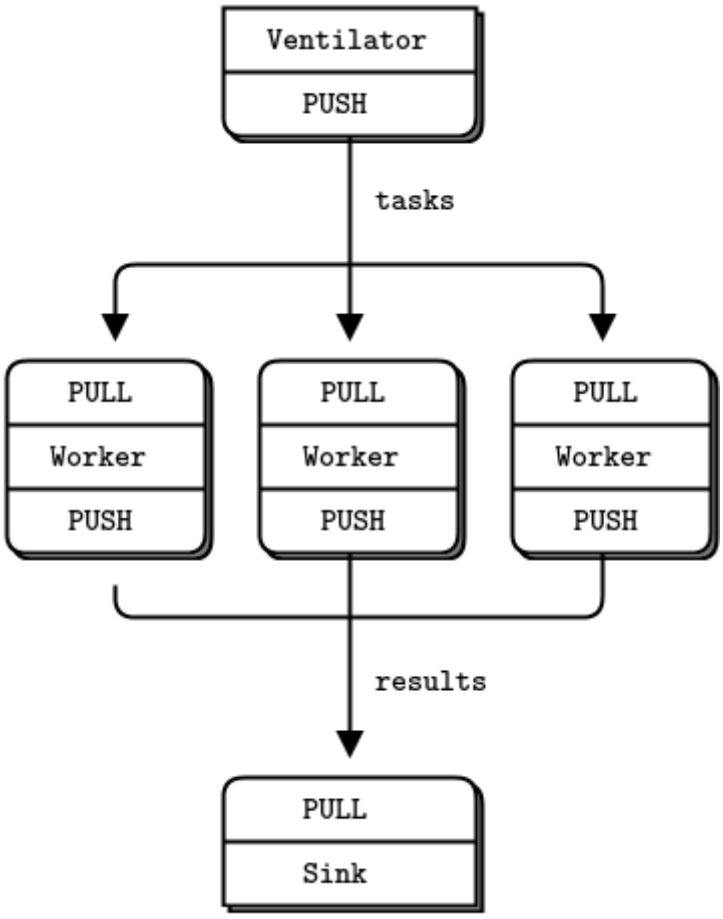
===

PULL/PUSH (Pipeline 패턴)



› 특징

- PUSH socket 은 연결이 존재하지 않을 때 send() 하기 되면 연결 될 때 까지 블록 된다.
- 전달된 메시지는 연결된 socket 에 round robin 된다.



- Worker는 ventilator에 PULL로 연결(connect)되어 있고, sink와는 PUSH로 연결(Connect) 되어 있다. 이것은 Worker 는 임의로 추가 할 수 있다는 것을 의미한다.
  만약 worker 가 바인딩(bind) 되어 있다면 worker 가 추가 될 때마다 매번 ventilator와 sink에 더

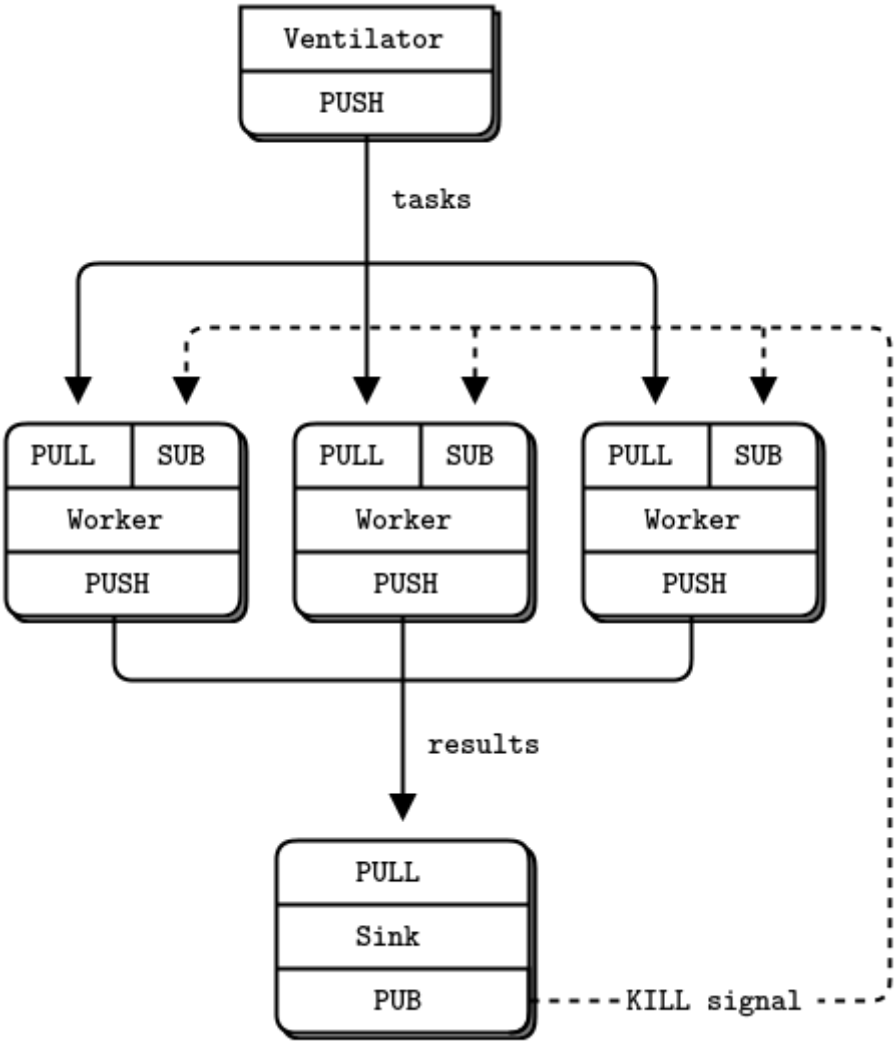
많은 소켓이 필요하다. 이 구조에서 ventilator와 sink 는 stable part, worker는 dynamic part 라 부른다.

- ventilator의 PUSH 소켓은 균등하게 Worker에 작업을 분배한다. (load-balancing)
- Sink의 PULL 소켓은 균등하게 Worker로 부터 결과를 수집한다. (fair-queuing)

PUSH, PULL의 bind, connect 는 상황에 따라 유용한 패턴이 있다.

- PUSH - bind, PULL - connect 의 경우는 동시 처리를 위한 Producer-Consumer 패턴에 적합
- PUSH - connect, PULL - bind 의 경우 처리 데이터를 한곳으로 집중 시켜 모을 때 좋음

Parallel Pipeline with Kill Signaling



Parallel task ventilator

3 초에 한번 메시지 전송

```
import zmq
import random
import time

context = zmq.Context()
sender = context.socket(zmq.PUSH)
sender.bind('tcp://127.0.0.1:10102')
```

```
sink = context.socket(zmq.PUSH)
sink.connect('tcp://127.0.0.1:10103')

print("Press Enter when the workers are ready: ")
_ = input()
print("Sending tasks to workers")

# The first message is "0" and signals start of batch
sink.send(b'0')

# Initialize random number generator
random.seed()

# Send 100 tasks
total_msec = 0

for task_nbr in range(100):
    # Random workload from 1 to 100 msecs
    workload = random.randint(1, 100)
    total_msec += workload

    sender.send_string(u'%i' % workload)
    print(i)

print("Total expected cost: %s msec" % total_msec)

# Give 0MQ time to deliver
time.sleep(1)
```

Parallel task worker

```
import sys
import time
import zmq

context = zmq.Context()

# Socket to receive messages on
receiver = context.socket(zmq.PULL)
receiver.connect("tcp://localhost:10102")

# Socket to send messages to
sender = context.socket(zmq.PUSH)
sender.connect("tcp://localhost:10103")

# Process tasks forever
while True:
    s = receiver.recv()

    # Simple progress indicator for the viewer
    sys.stdout.write('.')
    sys.stdout.flush()
```

```
    # Do the work
    time.sleep(int(s)*0.001)

    # Send results to sink
    sender.send(b'')
```

Parallel task sink

```
import sys
import time
import zmq

context = zmq.Context()

# Socket to receive messages on
receiver = context.socket(zmq.PULL)
receiver.bind("tcp://127.0.0.1:10103")

# Wait for start of batch
s = receiver.recv()

# Start our clock now
tstart = time.time()

# Process 100 confirmations
total_msec = 0
for task_nbr in range(100):
    s = receiver.recv()
    if task_nbr % 10 == 0:
        sys.stdout.write(':')
    else:
        sys.stdout.write('.')
    sys.stdout.flush()

# Calculate and report duration of batch
tend = time.time()
print("Total elapsed time: %d msec" % ((tend-tstart)*1000))
```

context.term() 을 호출 하기 전에 체크 사항

- 열려있는 소켓이 있다면 Blocking (LINGER : 0 시에도 Blocking)
- 소켓이 Close() 상태더라도 send() 가 처리 되기 전까지 Blocking (LINGER : 0 적용시 예외)
- 즉 데이터 전송이 완료된 이후에 소켓 종료, Context 종료 순으로 진행되어야 함. 데이터 전송과 상관없이 종료 하고 싶으면 LINGER zero 옵션 적용

Handling Multiple Sockets

This version uses a simple recv loop

```
import zmq
import time

# Prepare our context and sockets
context = zmq.Context()

# Connect to task ventilator
receiver = context.socket(zmq.PULL)
receiver.connect("tcp://localhost:5557")

# Connect to weather server
subscriber = context.socket(zmq.SUB)
subscriber.connect("tcp://localhost:5556")
subscriber.setsockopt(zmq.SUBSCRIBE, b"10001")

# Process messages from both sockets
# We prioritize traffic from the task ventilator
while True:

    # Process any waiting tasks
    while True:
        try:
            msg = receiver.recv(zmq.DONTWAIT)
        except zmq.Again:
            break
        # process task

    # Process any waiting weather updates
    while True:
        try:
            msg = subscriber.recv(zmq.DONTWAIT)
        except zmq.Again:
            break
        # process weather update

    # No activity, so sleep for 1 msec
    time.sleep(0.001)
```

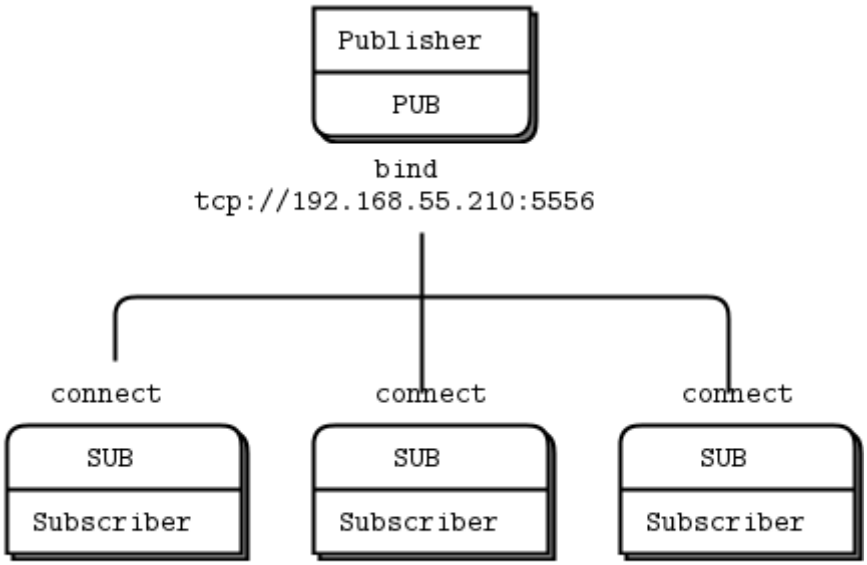
This version uses zmq.Poller()

```
import zmq

# Prepare our context and sockets
context = zmq.Context()

# Connect to task ventilator
receiver = context.socket(zmq.PULL)
receiver.connect("tcp://localhost:5557")
```

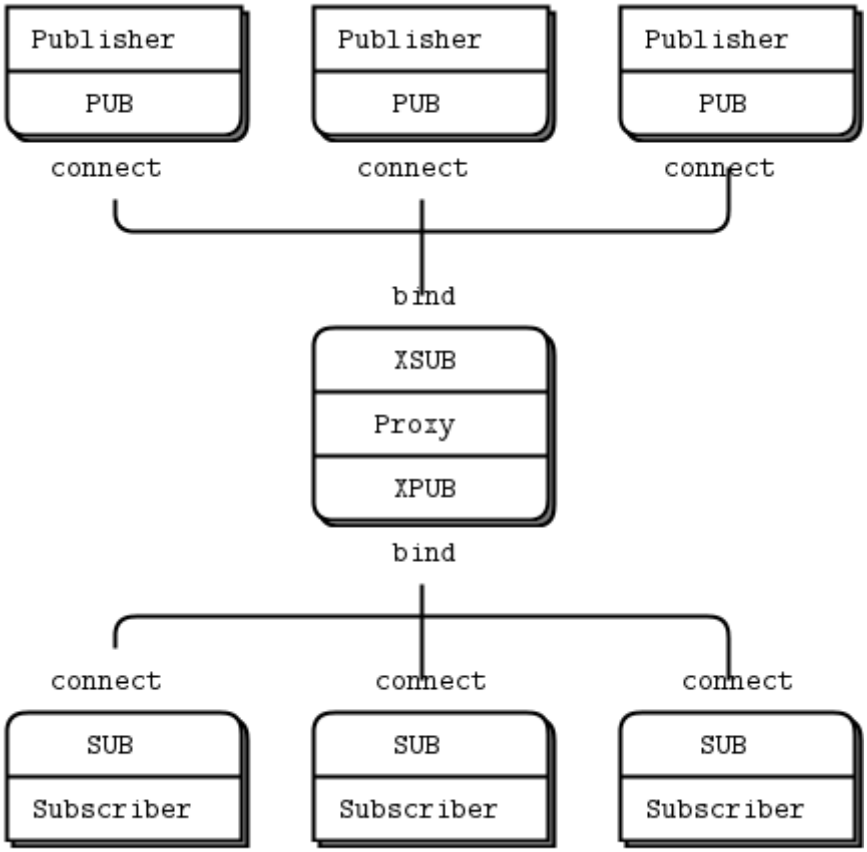
```
    # Connect to weather server
    subscriber = context.socket(zmq.SUB)
    subscriber.connect("tcp://localhost:5556")
    subscriber.setsockopt(zmq.SUBSCRIBE, b"10001")

    # Initialize poll set
    poller = zmq.Poller()
    poller.register(receiver, zmq.POLLIN)
    poller.register(subscriber, zmq.POLLIN)

    # Process messages from both sockets
    while True:
        try:
            socks = dict(poller.poll())
        except KeyboardInterrupt:
            break

        if receiver in socks:
            message = receiver.recv()
            # process task

        if subscriber in socks:
            message = subscriber.recv()
            # process weather update
```

그밖에 메시지 패턴

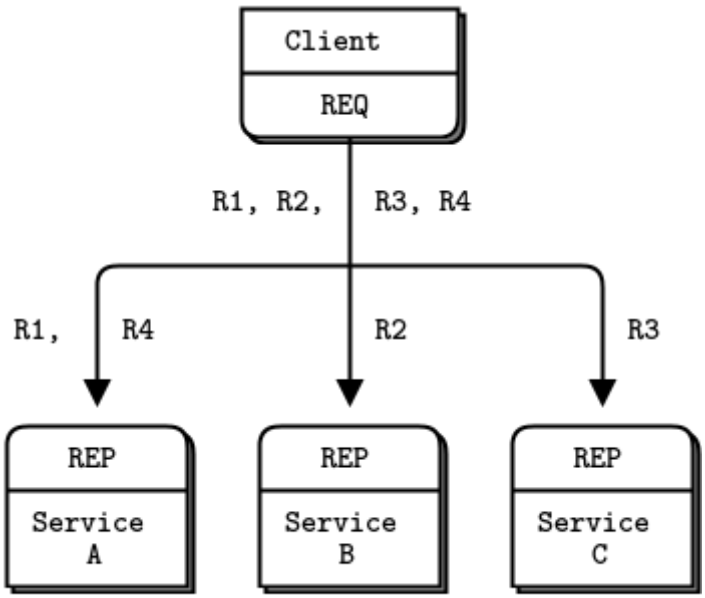
Small-Scale Pub-Sub Network



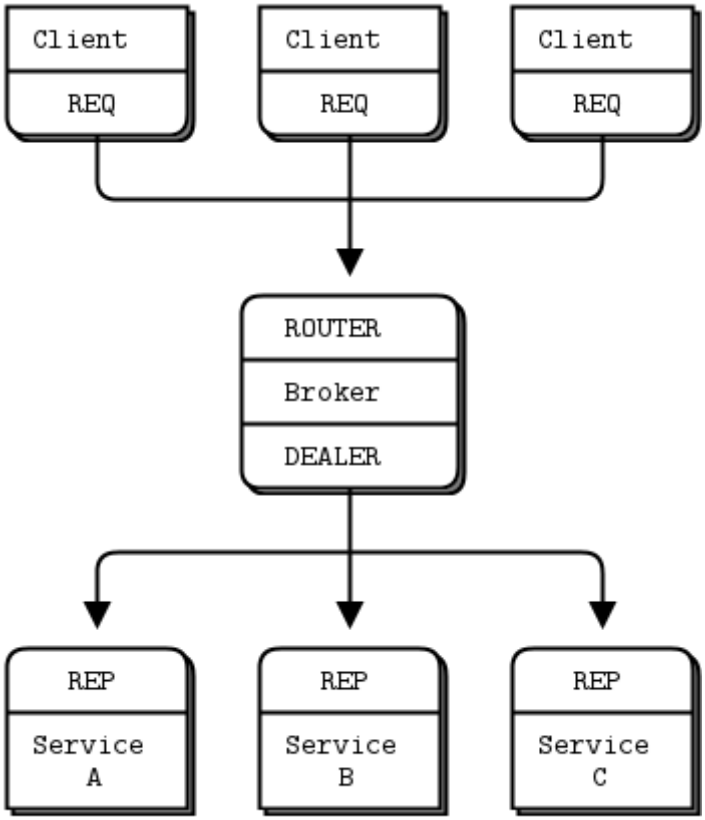
Extended Pub-Sub



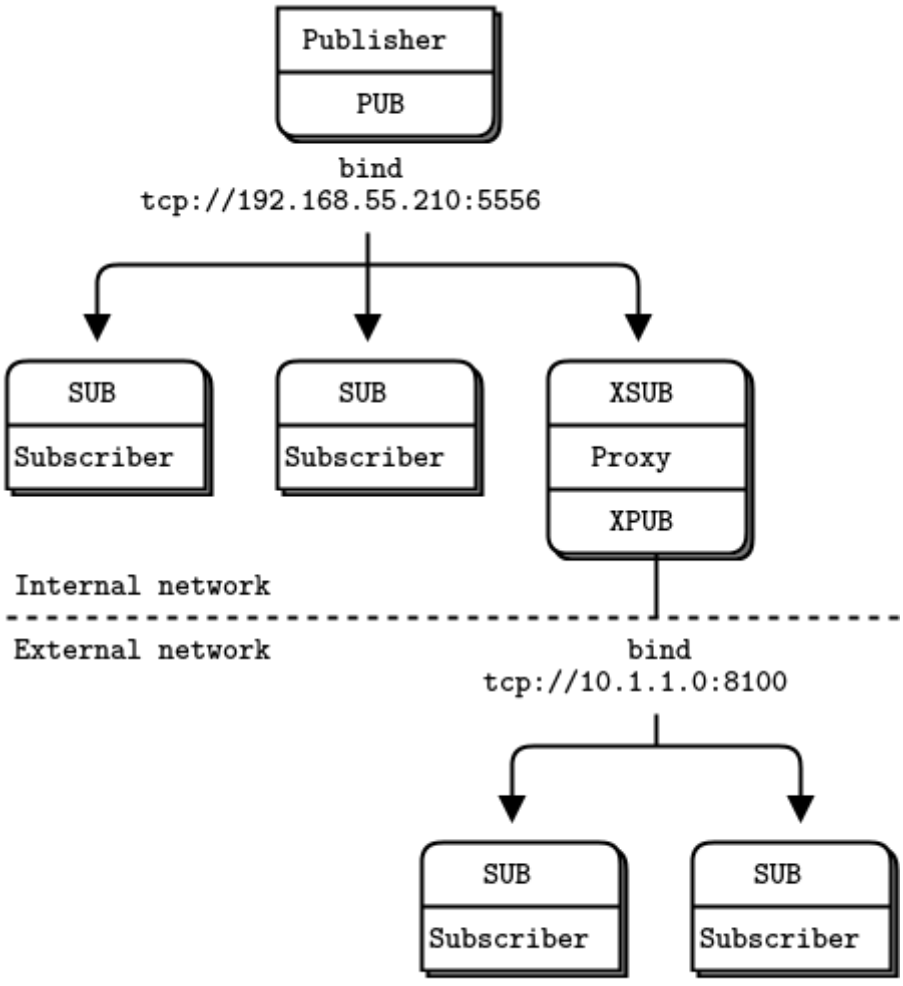
Request Distribution



Extended Request-Reply



Pub-Sub Forwarder Proxy



Queue Broker Exameple

Simple request-reply broker

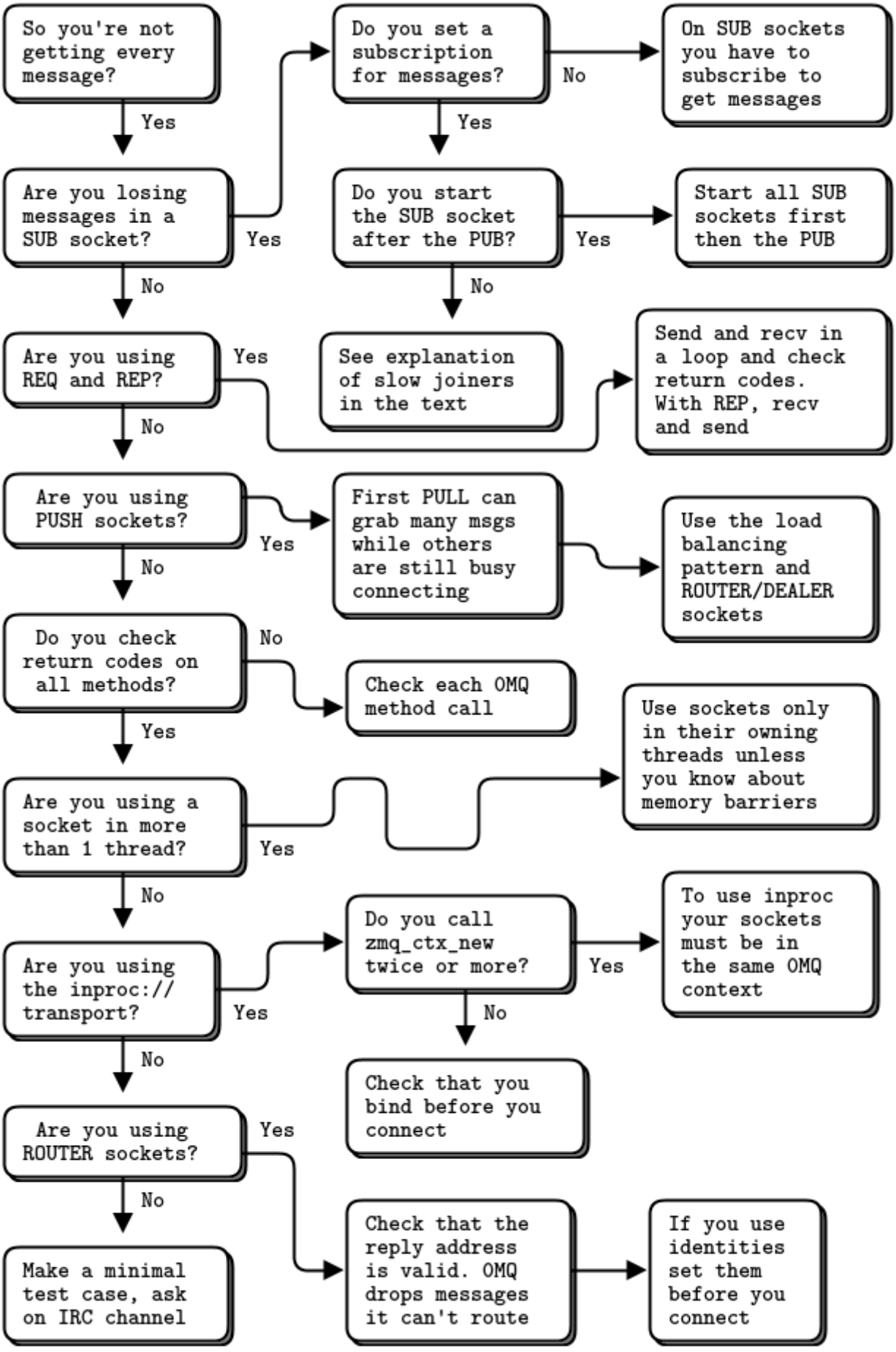
```
import zmq

# Prepare our context and sockets
context = zmq.Context()
frontend = context.socket(zmq.ROUTER)
backend = context.socket(zmq.DEALER)
frontend.bind("tcp://*:5559")
backend.bind("tcp://*:5560")

# Initialize poll set
poller = zmq.Poller()
poller.register(frontend, zmq.POLLIN)
poller.register(backend, zmq.POLLIN)

# Switch messages between sockets
while True:
    socks = dict(poller.poll())

    if socks.get(frontend) == zmq.POLLIN:
        message = frontend.recv_multipart()
        backend.send_multipart(message)

    if socks.get(backend) == zmq.POLLIN:
        message = backend.recv_multipart()
        frontend.send_multipart(message)
```

Same as request-reply broker but using zmq.proxy

```
import zmq

def main():
    """ main method """

    context = zmq.Context()

    # Socket facing clients
    frontend = context.socket(zmq.ROUTER)
    frontend.bind("tcp://*:5559")

    # Socket facing services
    backend  = context.socket(zmq.DEALER)
    backend.bind("tcp://*:5560")

    zmq.proxy(frontend, backend)

    # We never get here…
    frontend.close()
```

```
        backend.close()
        context.term()

    if __name__ == "__main__":
        main()
```

Missing Message Problem Solver


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

참고자료

- Euro PyCON 2011 발표자료
- zeromq 한글 번역
- 파이썬 zeromq 가이드