ZeroMQ at home

Jean-Claude Wippler
jeelabs.org

The bigger picture

How to develop software that lasts

Data, not code & state, not a database

Fail-safety, robustness, resilience

A decentralised design for the home

Software that lasts

Aim for city-like evolution

Do **not** pick one operating system

Do **not** pick one programming language

Do **not** pick one database format

Do pick an inter-connect / messaging system

Do pick a message protocol format

More decisions

Libraries vs frameworks

Focus on data structures, not code

Because the data will be with you forever!

The essence of data is **not** the database

The essence is a **model** of your home

State

The model is: the state of your home

temperatures and other sensor values

current (or recent) trigger events

light switches and dimmer settings

A kilobyte of data, that's probably it!

Wait...

Wasn't this supposed to be a talk about

ZeroMQ ?

Yes, but only after we see the context!

What is ZeroMQ?

A “better” low-level communication stack
Pub/sub, fan-in, fan-out, parallel/sequential
Fail-safe/-over, redundancy, network topology
Content is opaque: N bytes, or a bunch of ‘em

But addressing is made very explicit
Put stuff inside an envelope or open it up
Elegance: route pushed in front of the payload
Portable: bindings for dozens of languages
Excellent performance when used properly
Still small enough to fit on a tiny Linux box

Whoa... one step back

It's all about messages:

Sending, receiving, routing

Within a process, e.g. between threads

Between processes (and languages) ➙ IPC

Between machines ➙ sockets

An example

```
   import zmq
import time
context = zmq.Context()

subscriber = context.socket (zmq.SUB)
subscriber.connect ("tcp://192.168.55.112:5556")
subscriber.connect ("tcp://192.168.55.201:7721")
subscriber.setsockopt (zmq.SUBSCRIBE, "NASDAQ")

publisher = context.socket (zmq.PUB)
publisher.bind ("ipc://nasdaq-feed")

while True:
    message = subscriber.recv()
    publisher.send (message)
```


Design choices

Do I want to publish to N? → PUB / SUB

Do I want to drive N in parallel? → PUSH

Do I want to dispatch to N? → ROUTE

Synchronous → request / reply

Asynchronous → send + call-back function

Message content is irrelevant → plumbing!

More decisions

End points - either side can start first

Message broker to collect / dispatch work
need not be application-specific

Make key components state-less if possible
fail-safe, fail-over, resilient, redundant

In the home: decentralised topology?
run subsystems on batteries or solar

Why ZeroMQ?

It forces you to think about the BIG picture

No lock-in: code, data, language, OSS

Topology can be adjusted over time

3 P's → Pipes • Plumbing • Process

3 S's → Simple • Solid • Scalable

The flip side

You have to start from scratch

You don't get a message protocol

You need to implement fail-safety

It's going to be a lot of work!

Step One

Big decisions:

- Decide to base everything on ZeroMQ
- Use Bencode as message format

Small decisions:

- Use (embedded) Linux as platform
- Use Lua as programming language

Steps 2 .. N

Must read:

www.zeromq.org

zguide2.zeromq.org/page:all

My weblog is at jeelabs.org

jeelabs.org/2012/06/22/structured-data/