# What is ZeroMQ?

The ØMQ guide starts off with a section called ‘ZeroMQ in a Hundred Words’. This paragraph I think does a great job at describing what it is to someone who is experienced in network programming and distributed systems, but to someone not so experienced, it is a hundred words of mostly jargon. So here’s my attempt at ZeroMQ in a hundred words or less.

### ****ZeroMQ in a Hundred Words****

ZeroMQ is an asynchronous network messaging library known for its high performance. It’s intended use is for distributed systems as well as concurrent systems. In summary, ZMQ allows you to send messages (binary data, serialized data, simple strings, etc.) over the network through various methods like TCP or multicast as well as between processes.  ZeroMQ provides a whole slew of language APIs which run on most operating systems and allows you to communicate seamlessly between all sorts of programs. It also provides a collection of patterns, such as request-reply and publish-subscribe which assist you in creating and structuring your network.

### ****Sockets****

Sockets are the bread and butter of ZeroMQ. ZMQ contains several different sockets, each with their own properties and use cases. Sockets can be combined in many different ways, though there are plenty of combinations of sockets that are simply incompatible. Understanding the basics of sockets is key to designing your ZeroMQ system.

I’ll be running through a few of the most of the socket types and some of their typical combinations, though there are many different socket combinations other than the ones that will be discussed here.

### ****REQ and REP Socket****

The REQ socket is short for ‘request’. This is one of the most basic sockets that is typically seen paired with a REP, or ‘response’, socket. Though, the REQ socket as well as the REP socket can be combined with many other types of sockets.

The REQ/REP socket combination works something like this:

1. The REQ and REP sockets are initialized in separate programs. We call the “connect()” function on the REQ socket and “bind()” on the REP socket.
2. The REP socket is initialized and is blocked, waiting for a message to arrive.
3. The REQ socket sends some arbitrary “request” over the network
4. REP receives this message and does whatever it needs to do with it, and the REP socket can send a reply/acknowledgement back to the REQ socket.

Don’t worry about connect vs. bind right now, but if you want to learn more, check out the “*When should I use bind and when connect?*” question in the FAQ.

Keep in mind, the REQ/REP sockets are *synchronous* sockets. This means that they can only talk to one peer at a time. Because of this, the use case for the REP/REQ socket combination is very narrow and you typically won’t find it in a real world application. You should really only be using REQ/REP as a learning tool to help you understand the basics of ZeroMQ

### ****ROUTER and DEALER Socket****

The DEALER-ROUTER sockets can be thought of as a non-blocking, asynchronous version of the REQ-REP sockets. The DEALER socket acts as the REQ socket and the ROUTER acts as the REP socket. Something to keep in mind while researching ZMQ on your own, you’ll notice in older blogs/posts the term XREQ and XREP. These are the old names for the DEALER and ROUTER sockets.

Sending and receiving from a ROUTER socket is a little different from other sockets. These differences allow it to be asynchronous. The most important thing to note is that a ROUTER socket expects any incoming messages to have an identity frame prepended to the message itself. This allows the ROUTER to know where the message came from. When sending a message on a ROUTER, the first frame of the message is removed, and used to identify which client to send the response to.

The DEALER sockets are intended to ‘deal’ messages out to its connected workers. When sending a message out via the DEALER socket, messages are sent in a ‘round-robin’ fashion.

To gain a more in depth understanding of these sockets, as well as other sockets involving the request/reply structure, I highly recommend The Request-Reply Mechanisms section of the ZeroMQ guide.

### ****PUB and SUB Sockets****

PUB/SUB stands for, as you might have guessed, ‘publish’ and ‘subscribe’. The way this socket combo works may be clearly evident to some, but I’d like to briefly go over it in case you aren’t familiar with the publish/subscribe model.

The basic idea of a publish/subscribe model is that a PUB socket pushes messages out, and all the associated SUB sockets receive these messages. This communication is strictly one way, there are no responses or acknowledgements sent by the SUB sockets.

### ****PAIR Sockets****

PAIR sockets are a socket type that can only be used with each other. PAIR sockets allow bidirectional communication and can only be connected to one peer at a time.

One very important thing to note is that PAIR sockets *cannot* be used over the network. They can only be used for messaging *between process threads.*

### ****PUSH/PULL Sockets****

The PUSH/PULL sockets are a one way socket combination that allows you to distribute messages to many workers. A PUSH socket will distribute sent messages to its PULL clients evenly.

This pattern is useful in cases where you are doing a large number of small tasks.

**How to install ZeroMQ on Ubuntu**

Before installing, make sure you have installed all the needed packages

**sudo apt-get install libtool pkg-config build-essential autoconf automake**

**sudo apt-get install libzmq-dev**

|  |
| --- |
| **# Build, check, and install libsodium**  **git clone git://github.com/jedisct1/libsodium.git**  **cd libsodium**  **./autogen.sh**  **./configure && make check**  **sudo make install**  **sudo ldconfig**  **cd ../**  **# Build, check, and install the latest version of ZeroMQ**  **git clone git://github.com/zeromq/libzmq.git**  **cd libzmq**  **./autogen.sh**  **./configure --with-libsodium && make**  **sudo make install**  **sudo ldconfig**  **cd ../**  # Now install ZMQPP  **git clone git://github.com/zeromq/zmqpp.git**  **cd zmqpp**  **make**  **make check**  **sudo make install**  **make installcheck** |

The make file can also build and install a client tool called zmqpp. To build this tool add the step

**make client**