comiit

Building distributed systems with ZeroMQ

Vladimir Smida
Software developer & Partner at Comiit ApS

12. August 2014

comrit

How do we fix the world?

comrit

How do we fix the world?

- First, we solve a general problem of

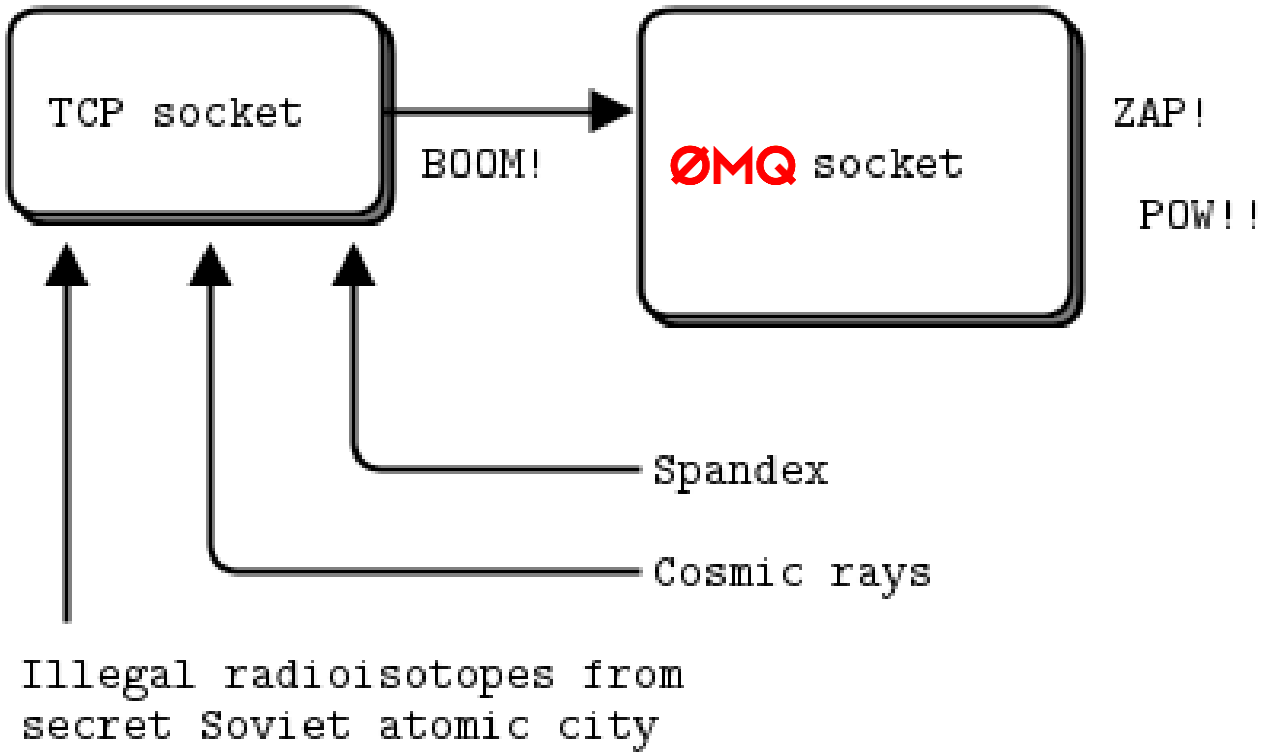
  “how to connect any code to any code, anywhere”

- Second, we wrap that up in the simplest possible building blocks that

  people could understand and use easily

comiit

Fixing the world…



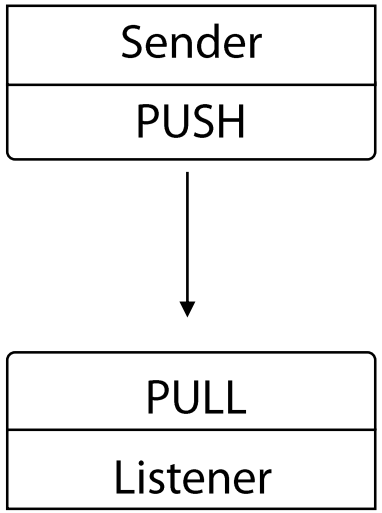
comiit

ZeroMQ

- Looks like an **embeddable networking library** but acts like a **concurrency framework**
- Based on low-level **socket API** but exposes **high-level messaging patterns**
- Gives you sockets that carry **atomic messages**
- **Open source** (LGPL)

comrit

Demo: Fire – and – Forget



comiit

Context

- Entry point
- **Single instance** in your process
  - knows about all sockets the process is using
  - inter thread communication
- carries configuration, disposes all sockets

Socket

- **Send and receive** messages
- Connects to/from **many endpoints**
- Defines communication pattern

comrit

Message

- ZeroMQ **transport messages** instead of stream of bytes

- **Not a neutral carrier**
  - Framing on the transport protocol
  - Not compatible with existing protocols (e.g. http)

- Message **body**
  - byte[]
  - String

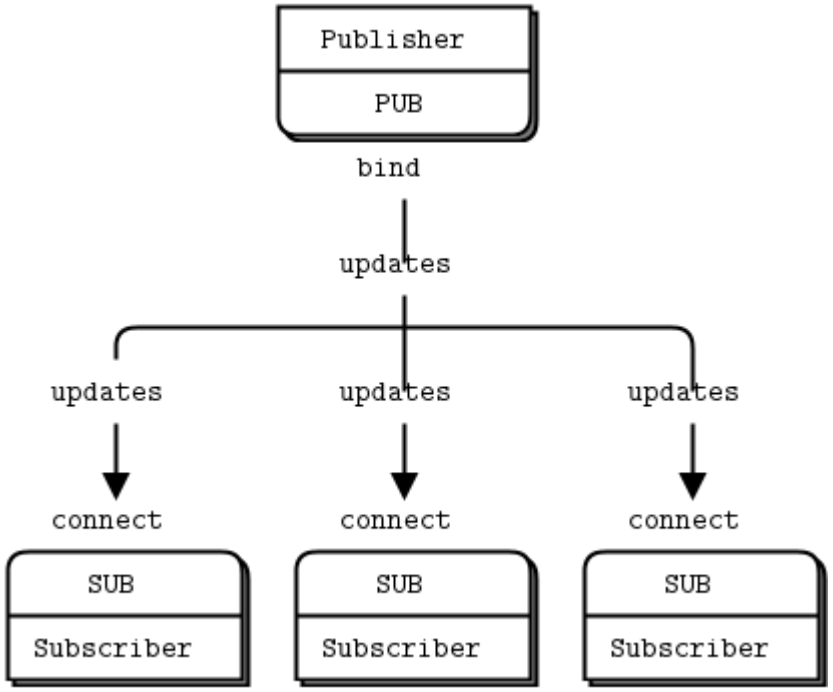
comrit

Asynchronous messaging

- Message **queueing**
- I/O in a **background thread**
- Socket can **connect** to endpoint **before** it has been **binded**
- Send() does not necessarily send message but **enqueue**

comrit

Demo: Publish - Subscribe



comiit

Publish - Subscribe

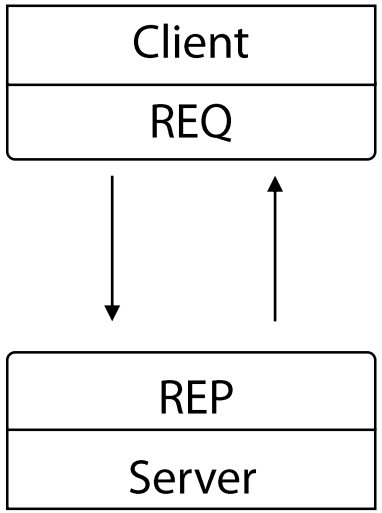
- subscriber can connect to **more than one publisher**
  - data will then arrive and be interleaved ("fair-queued")

- filtering happens at the **publisher side (**TCP, IPC)

- **Subscriber** receives messages only when connected
- **Publisher** drops **messages,** when there is **no subscriber**

comrit

Demo: Request - Reply



comrit

Simplicity

- No **serialization** or **compression**
- No **authentication** or **encryption**
- Reliable but **not durable** messaging

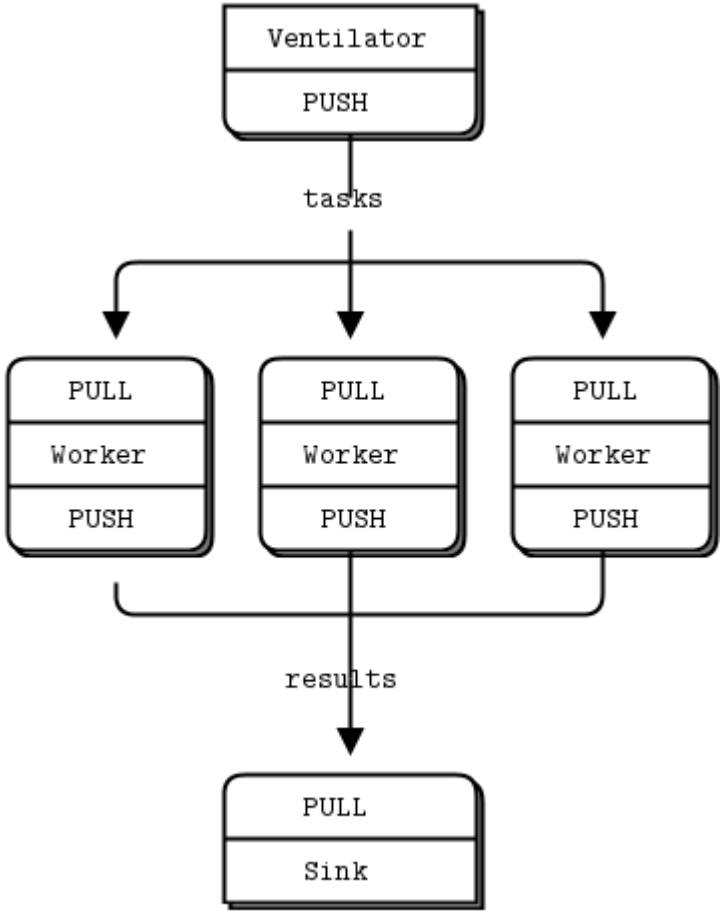
comrit

Cross - platform

- 30+ language bidings
  - Ada, Bash, Basic, C, ChickenScheme, CommonLisp, C#, C++, D, Erlang, F#, Felix, Flex(ActionScript), Go, Guile, Haskell, Haxe, Java, JavaScript(Flash), Lua, s, Node.js, Objective-C, ObjectiveCaml, ooc, Perl, PHP, Python, Racket, R, REBOL2, REBOL3, Red, Ruby, (FFI), Scala, Smalltalk, Tcl, Twisted(Python), …
- Interoperable messages
- OS-agnosticism: Linux, Windows, OS X
- Consistent API

comiit

Pipeline



comrit

ZeroMQ and .NET

- clrzmq
  - .NET ”binding” clrzmq.dll
  - open source, available on NuGet
  - bundled libzmq version - old (2.x) but stable
  - Build on top of native C++ library
    - libzmq.dll
    - current version 4.x
    - new transport protocol ZMTP provides cross platform **secure connection**
- NetMQ
  - Native .NET port

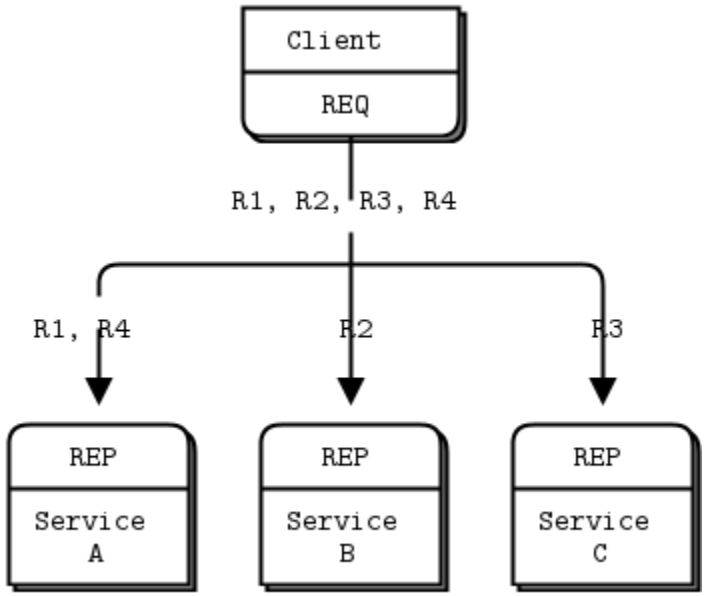
comrit

The Zen of Zero

- ØMQ
- Originally the zero in ØMQ was meant as **"zero broker"** and (as close to) **"zero latency"** (as possible)
- In the meantime it has come to cover different goals: zero administration, zero cost, zero waste
- More generally, "zero" refers to the culture of minimalism that permeates the project

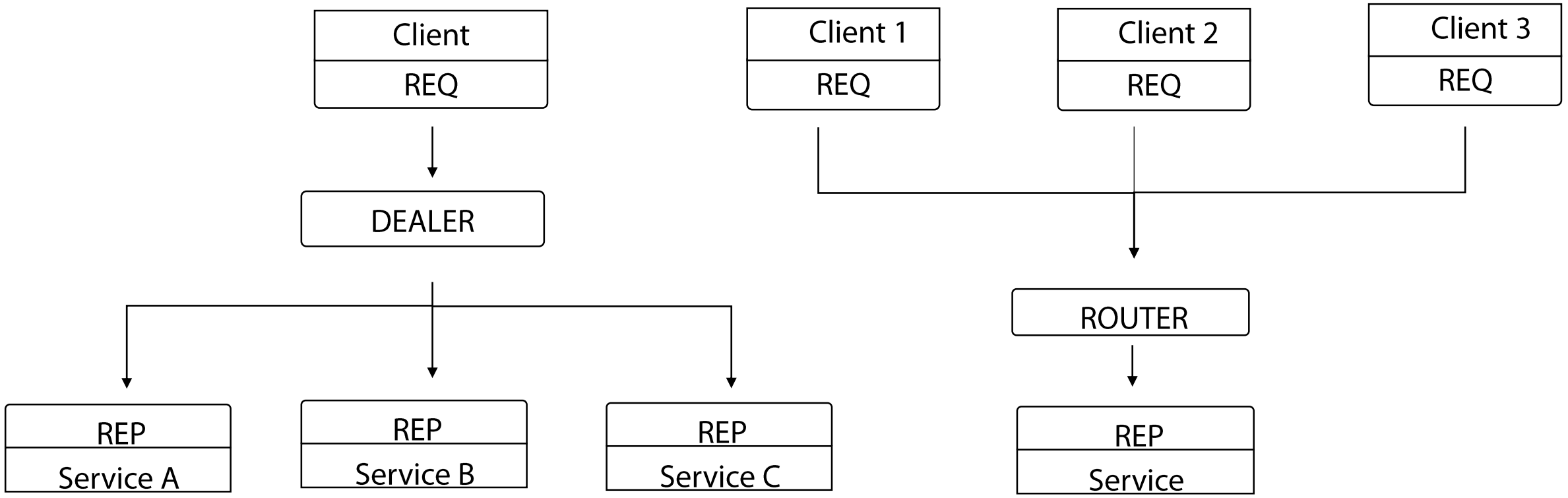
comiit

Router - Dealer



comrit

Router - Dealer



comrit

Network communication

- gives you **sockets that carry whole messages** across various transports
- You can **connect sockets N-to-N** with communication patterns

Multithreading

- Communication inside your host process
- we **don't need mutexes, locks**, or any other form of inter-thread communication **except messages sent across ØMQ sockets**

comrit

Different Transports

- INPROC:
  - **In-process** transport
  - passes messages **via memory** directly **between threads**
- IPC
  - **Inter-process** transport
  - passes messages **between local processes** using a **system-dependent IPC** mechanism
- TCP
  - ubiquitous, **reliable unicast** transport
- PGM/EPGM
  - protocol for **reliable multicast** transport of data over IP networks

comiit

Basic Buiding Blocks

Socket types

- Push
- Pull

- Pub
- Sub

- Req
- Rep

- Router
- Dealer

Messaging patterns

- Pipeline

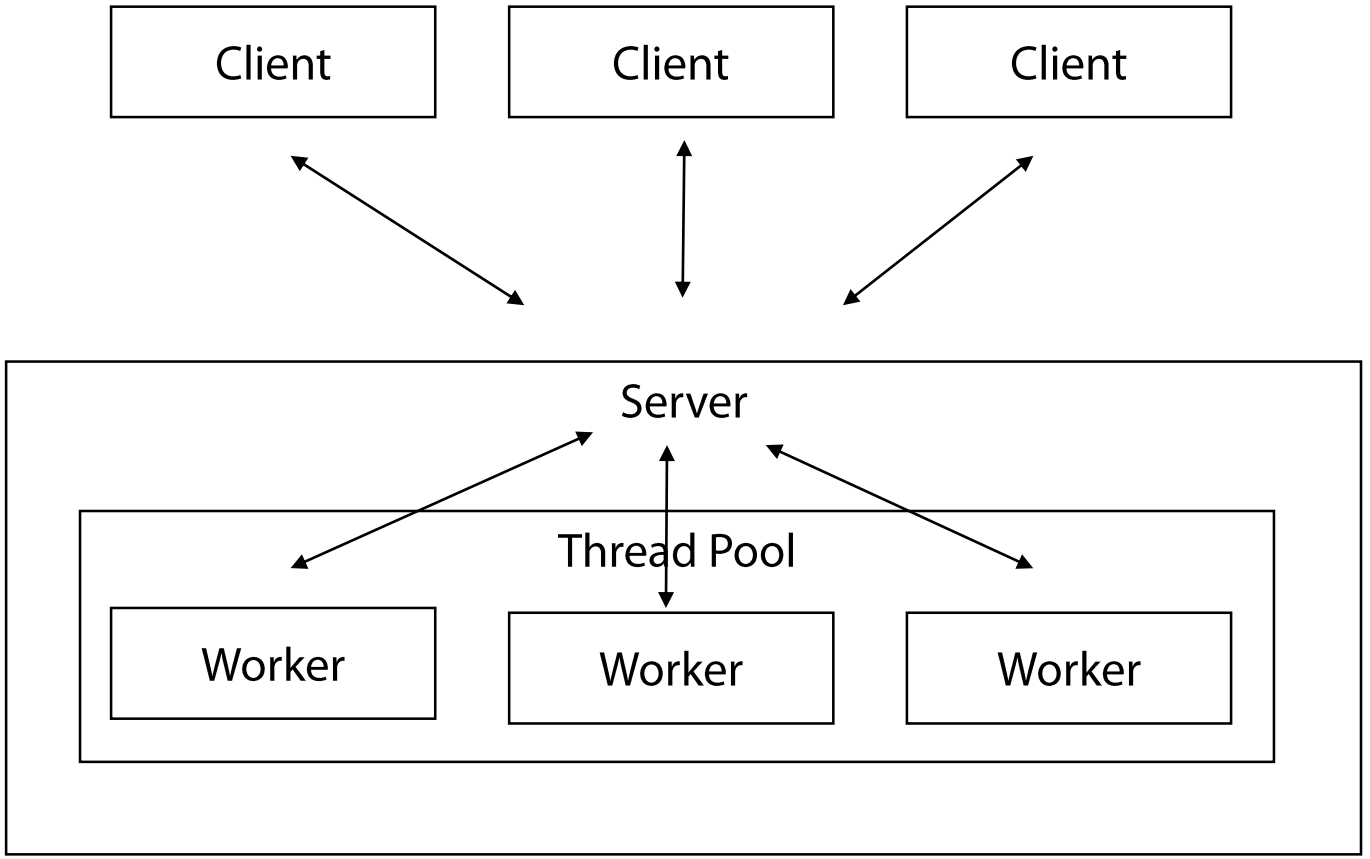
- Publisher – Subscriber

- Request – Reply

- (Exclusive pair)

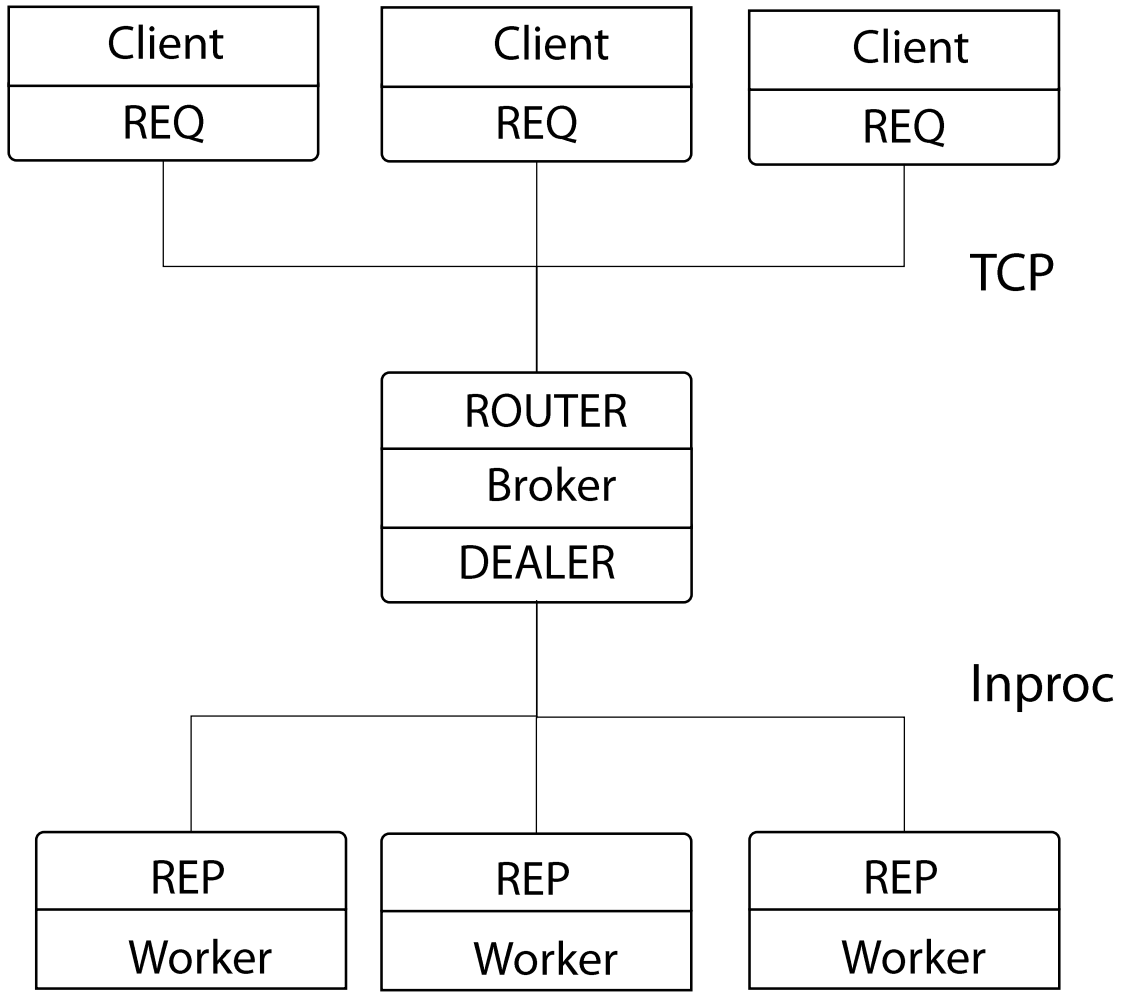
comrit

Load Balancer



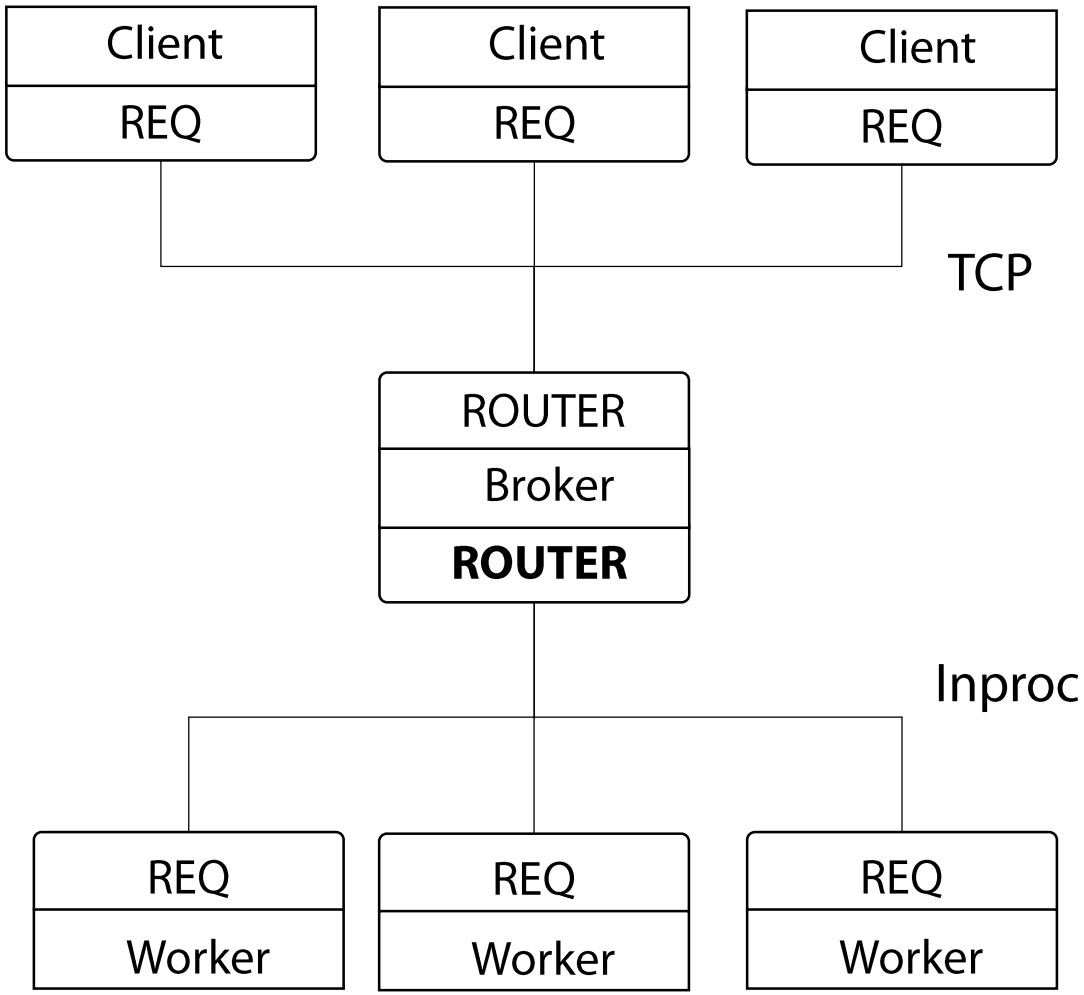
comrit

Demo: "Simple" Load Balancer



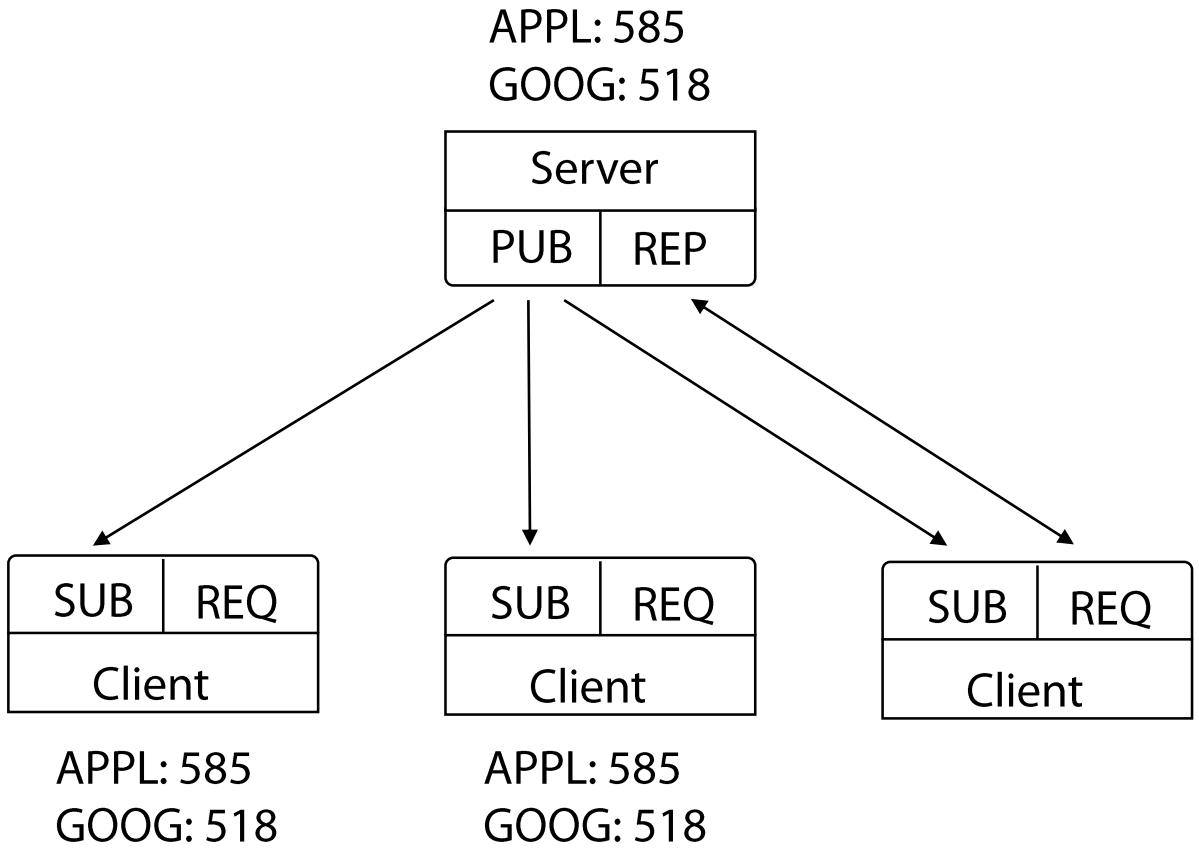
comrit

"Better" Load Balancer



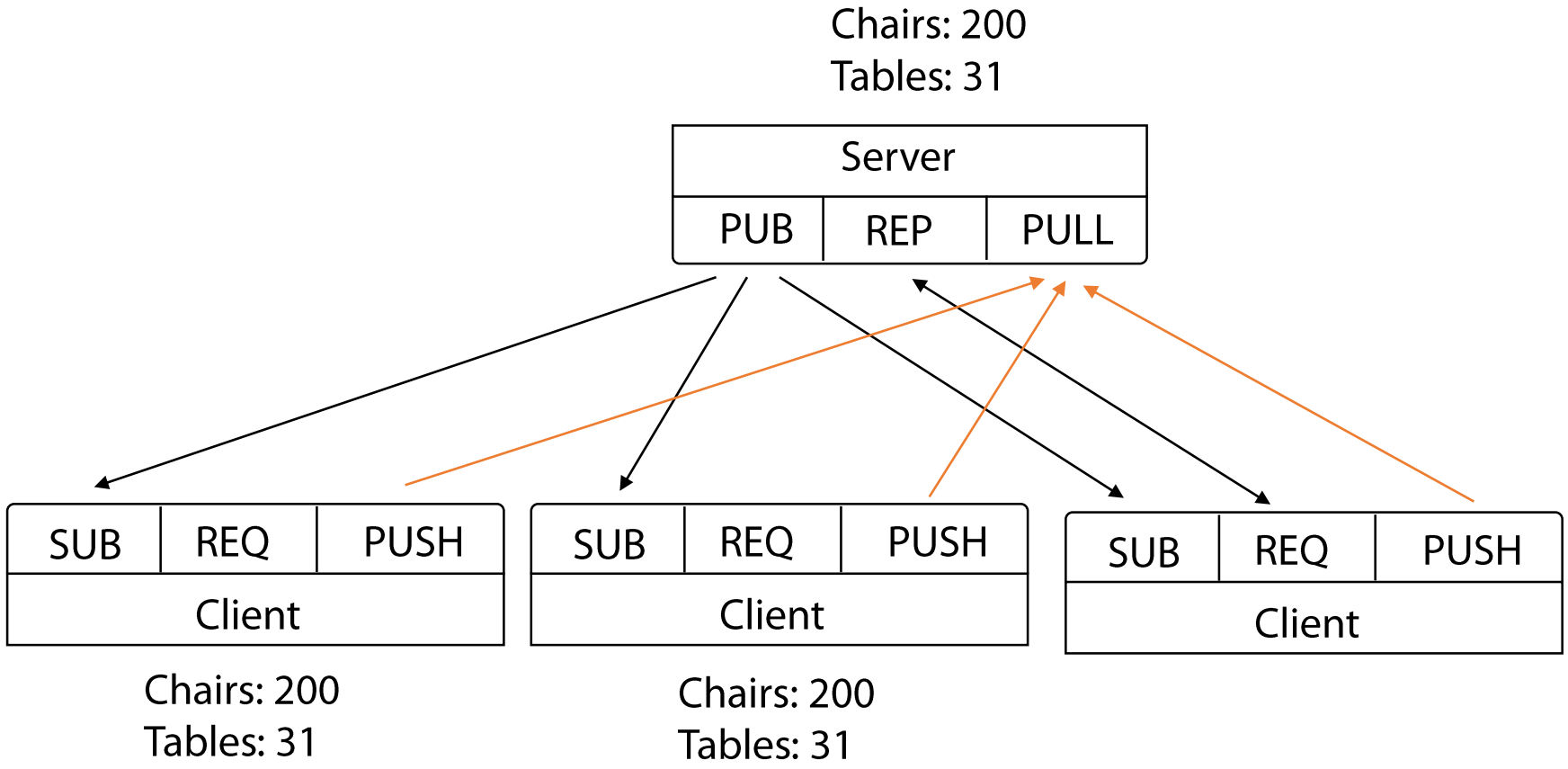
comrit

Publish - Subscribe with Snapshot



comrit

Publish-subscribe with snapshot and update



comiit

Reliability?

- Reliability
  - Crash of server
  - Crash of client
  - Unreliable network
  - Slow performance
  - Hardware can fail
  - Data centers can be struck by lightning, fire etc.
  - Informing user about disconnects
  - …
- How does it ZeroMq handle?

comiit

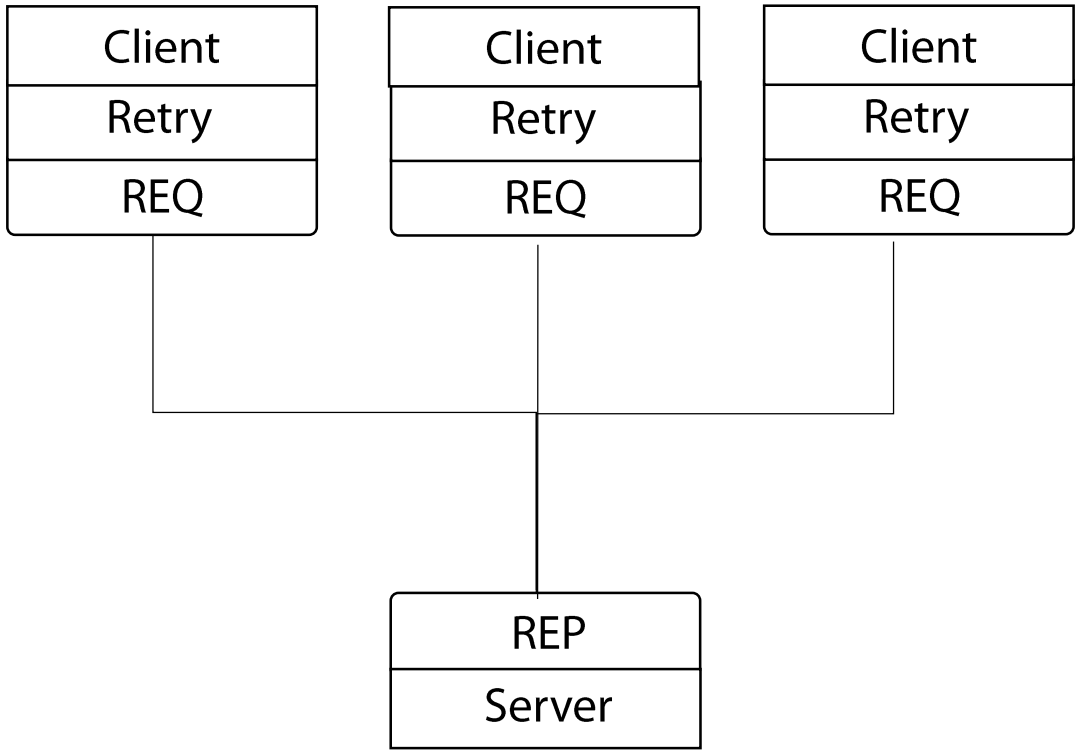
Reliability

- ZeroMQ does not handle all error cases but provides some tools for building reliability
- The level of reliability depends on the use-case

- Solutions
  - Retry on time-out
  - Heart beating
  - Limit on used resources
  - Idempotent
  - …

comrit

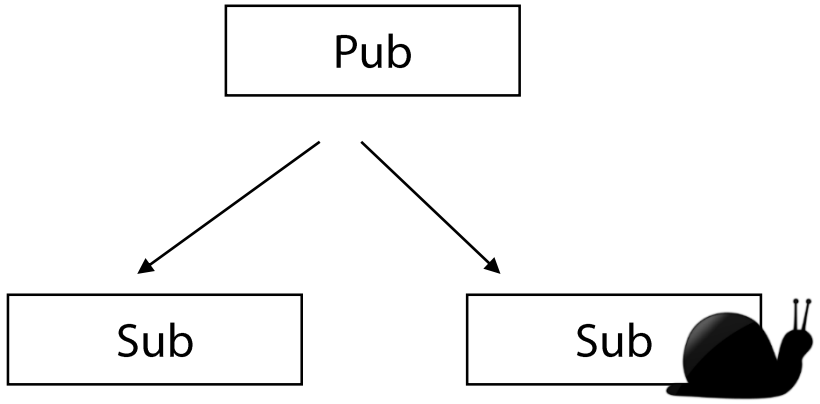
Lazy Pirate Pattern



comrit

Reliability: Slow subscriber

- What happens if there is a slow subscriber?

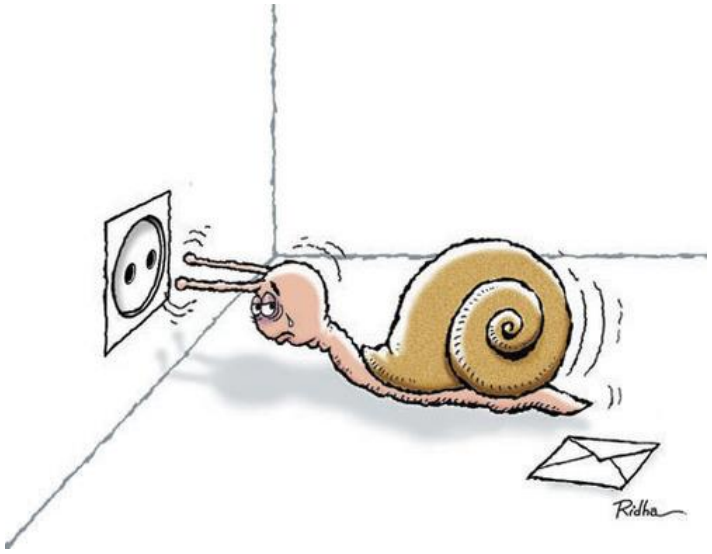


- Queue messages on the publisher?
- Queue messages on the subscriber?
- Stop queuing new messages after a while?
- Punish slow subscribers with disconnect?

comrit

Suicidal snail

- Subscriber receives messages from publisher
- Subscriber detects if it is receiving messages too slow
- If it receives too slow then it kills itself and thereby stop receiving messages



comrit

High water mark

- A way of dealing with over-full queues
- When a queue is full (reach it water mark) then it throws away new messages (or blocks sender)
- This means that queues will only contain a predefined amount of data
- However dropping messages might result in lost data

comrit

How to: Suicidal snail

- Put a high water mark on the publisher send queue
- Put a high water mark on the subscriber receive queue
- Sequence number on all messages being send
- When the subscriber receives the messages then it checks if the sequence numbers are consecutive
  - If they are not consecutive then messages has been lost because hitting the high water
  - The subscriber kills itself when messages are not consecutive

comrit

More reliability patterns

- Binary Star
- Paranoide Pirate
- Simple Pirate
- Majordomo
- Titanic
- Freelance
- …

comrit

Performance

- Throughput
- Latency

comiit

Performance

Model: Lenovo Z510

Operating System: Microsoft Windows 8.1
- Version: 6.3.9600

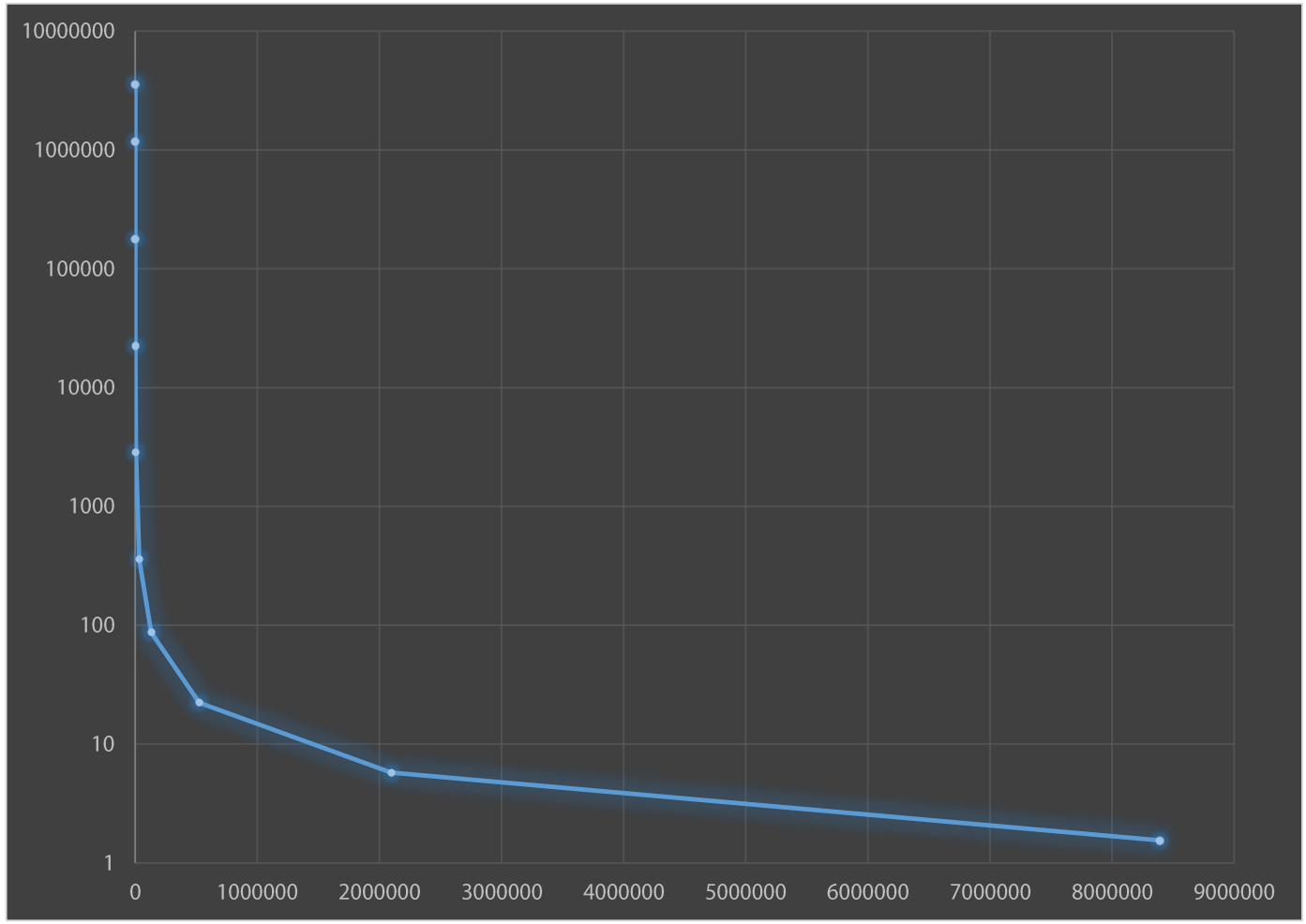
Network
- Realtek RTL8102E Family PCI-E Fast Ethernet (10/100MBit)

Number of Processors: 1
- Name: Intel 4 Core i7-4702MQ CPU @ 2.20GHz

Ram
- 16 GB

comrit

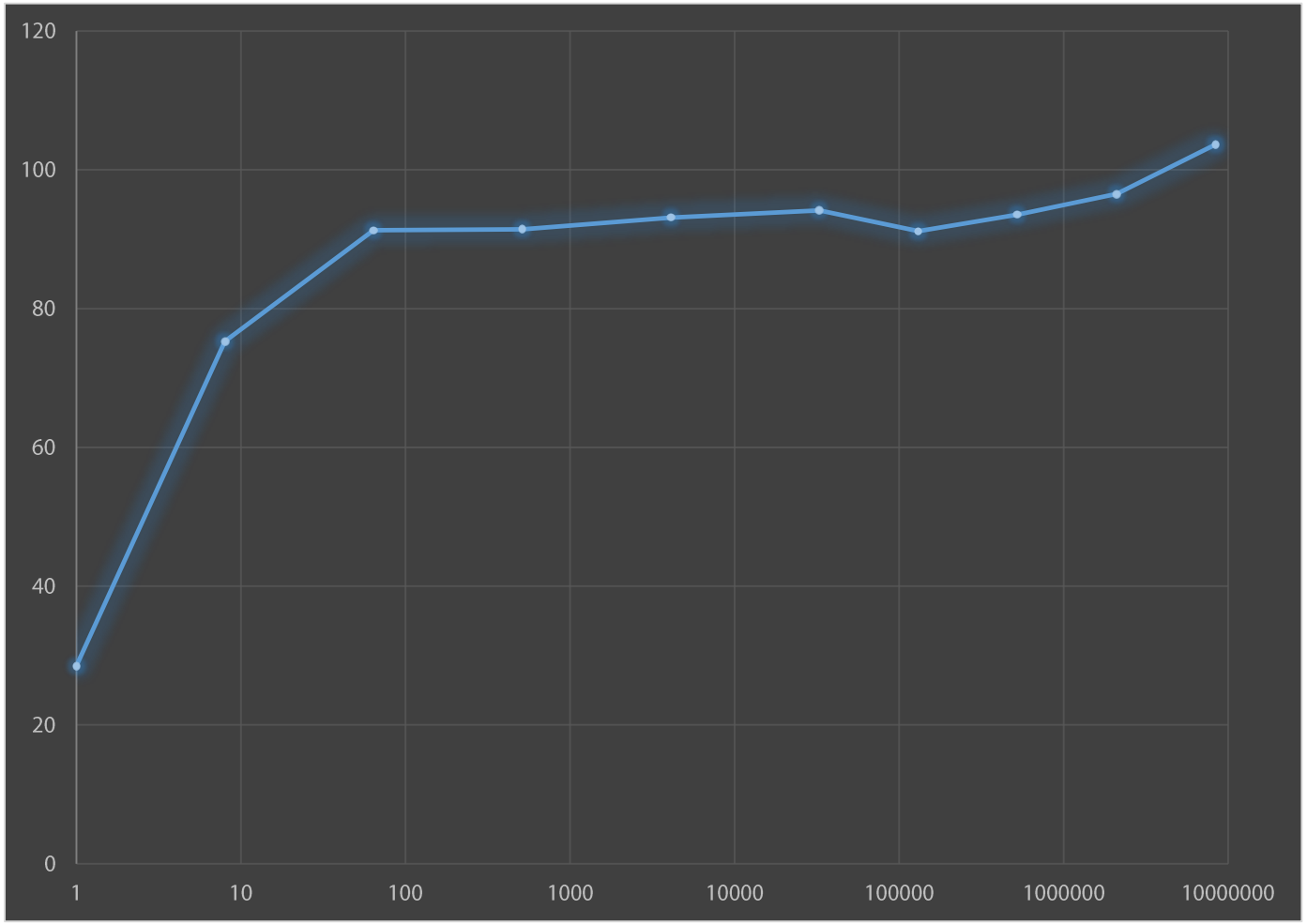
Remote: Messages per second



1 B	3562066
8 B	1176237
64 B	178276,2
512 B	22323,69
4 kB	2841,988
32 kB	359,1445
128 kB	86,9327
512 kB	22,3024
2 MB	5,752958
8 MB	1,544604

comrit

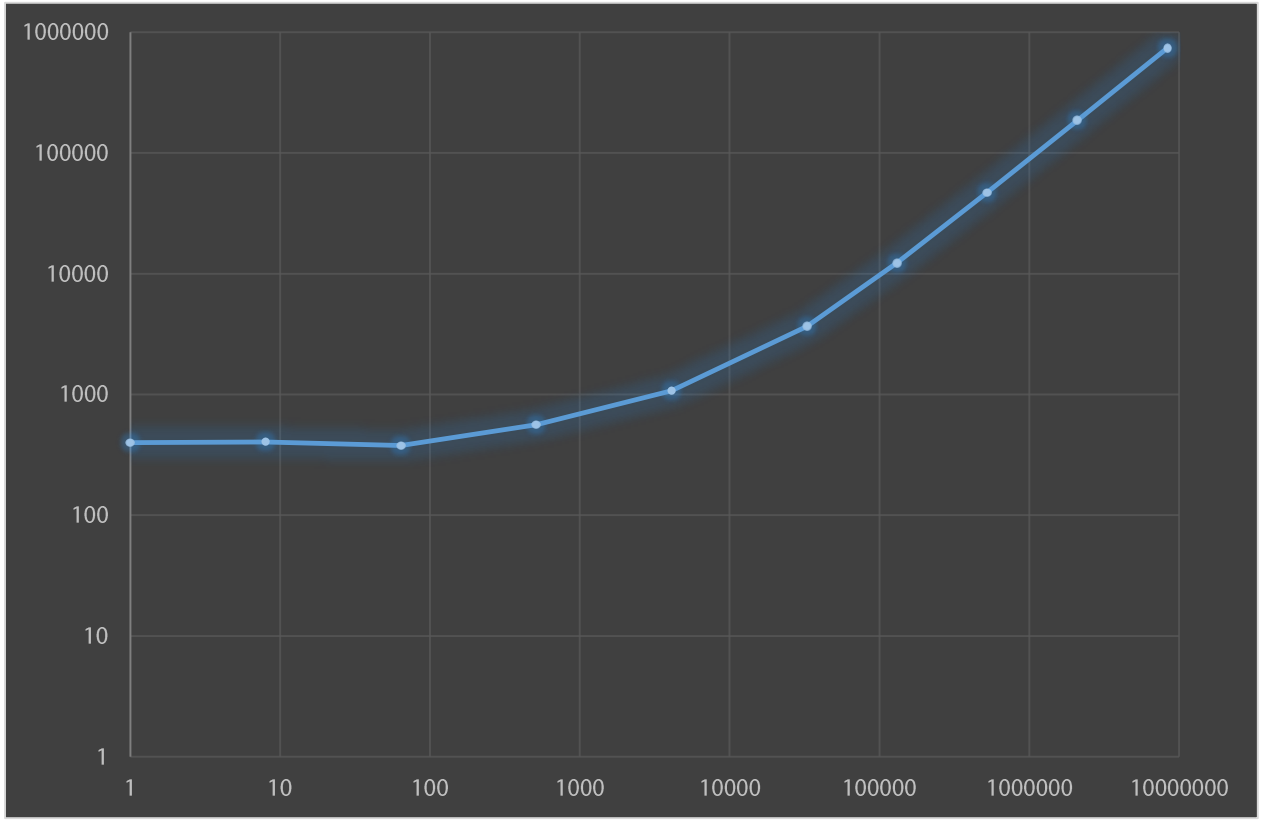
Remote: Mega bit per second



1 B	28,49652
8 B	75,27916
64 B	91,27742
512 B	91,43782
4 kB	93,12627
32 kB	94,14759
128 kB	91,15555
512 kB	93,54304
2 MB	96,51862
8 MB	103,6566

comiit

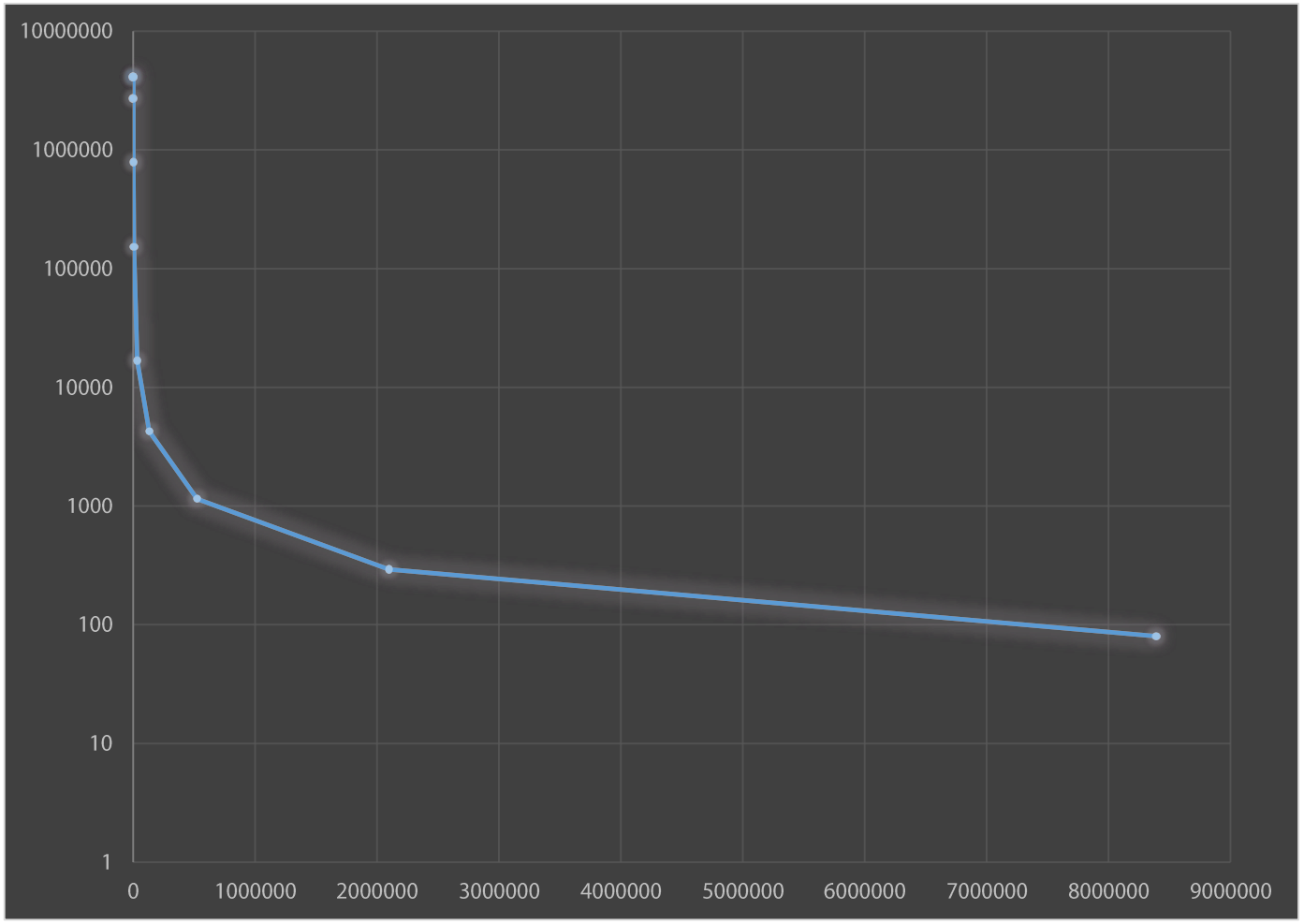
Remote: Latency (µs)



1 B	399,9246
8 B	404,4727
64 B	377,3626
512 B	562,872
4 kB	1073,362
32 kB	3668,614
128 kB	12305,67
512 kB	47084,2
2 MB	186710,4
8 MB	742378,3

comrit

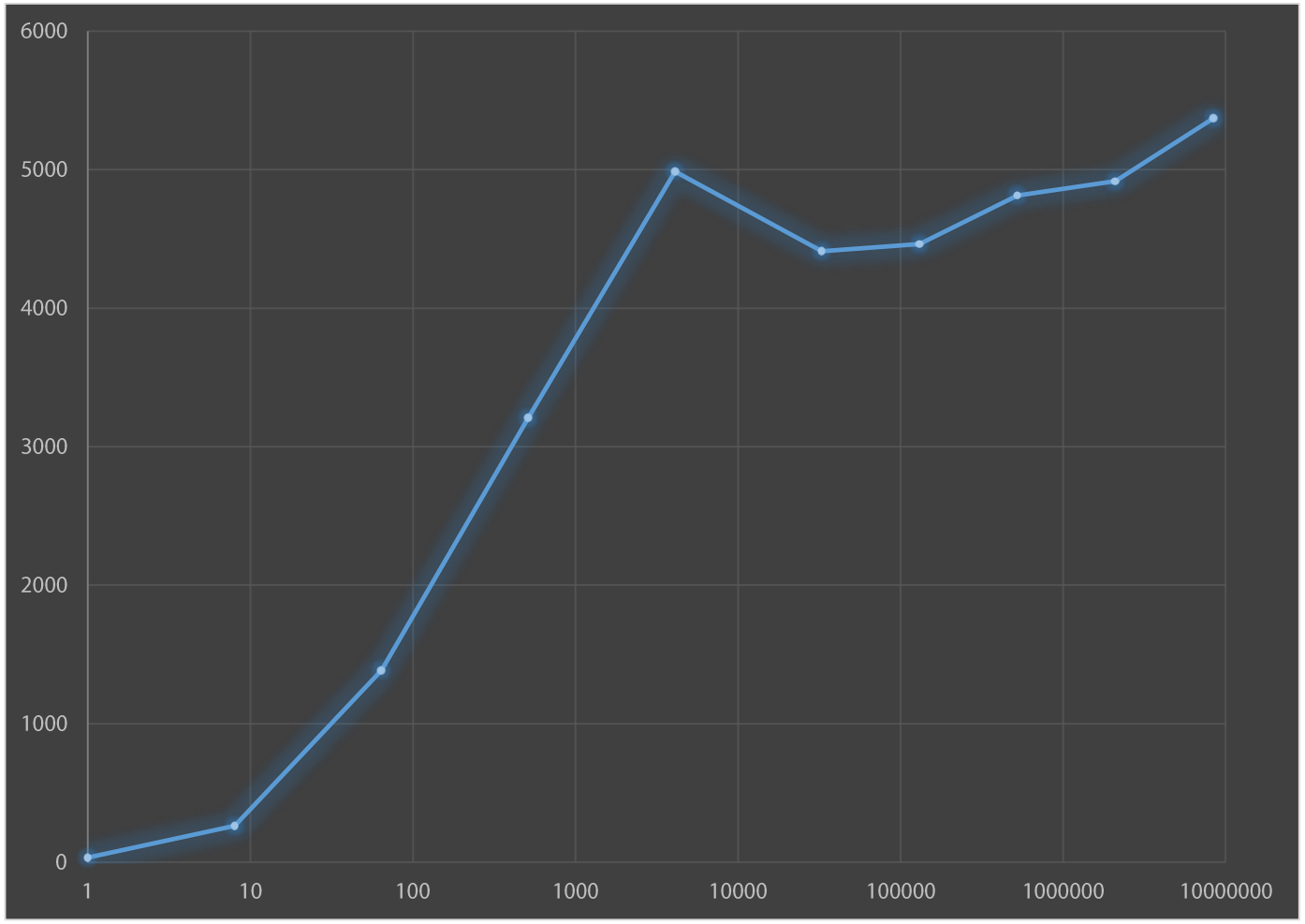
Localhost: Messages per second



1 B	4138129
8 B	4197820
64 B	2392897
512 B	819342
4 kB	150987
32 kB	16160
128 kB	5405
512 kB	1374
2 MB	327
8 MB	78

comit

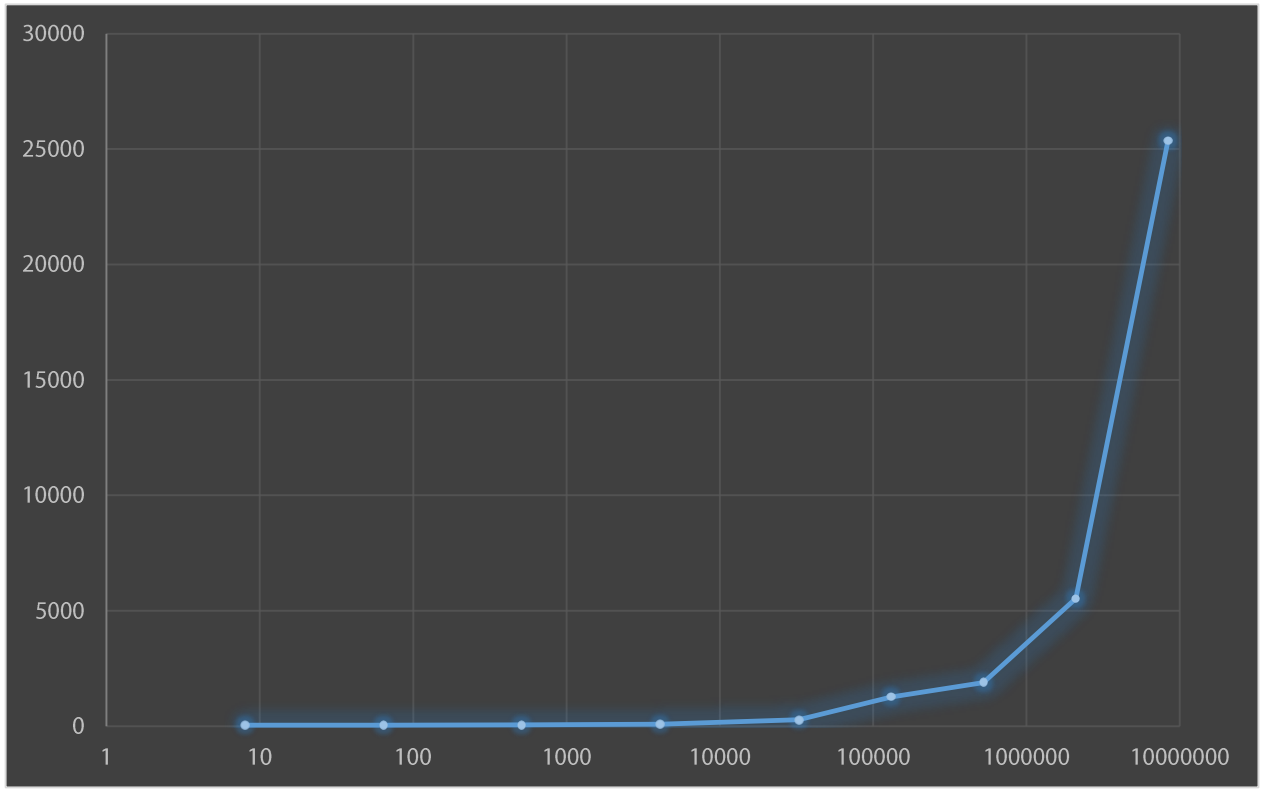
Localhost: Mega bit per second



1 B	33
8 B	268
64 B	1225
512 B	3356
4 kB	4947
32 kB	4236
128 kB	5667
512 kB	5762
2 MB	5486
8 MB	5234

comiit

Localhost: Latency (µs)



8	41,23854
64	42,54664
512	52,48148
4096	89,02846
32768	275,2133
131072	1264,997
524288	1888,259
2097152	5531,39
8388608	25366,24

comrit

Why ZeroMQ

- Open source
  - LGPL
  - Big active community
- High Performance
  - Send millions of messages per second
- “Easy” to use, integrate
  - Pleasure to code
- Flexible
  - Able to create all kinds of distributed systems
- Low running cost

comrit

Why not ZeroMQ

- Not a widely used protocol
  - So maybe not so good for public APIs as for example REST
- Custom protocols
  - E.g. cannot implement a web server
- Complexity
- Other communication frameworks exists for specific purposes
  - Web services
  - Message queue
  - ESB

comrit

Refferences

- http://zeromq.org/
- http://zguide.zeromq.org
  - Good guide to ZeroMQ
- https://github.com/zeromq/clrzmq
  - .Net binding
- http://pluralsight.com
- ZeroMQ: Messaging for Many Applications

comiit

Contact us

- Comiit ApS
  - Vesterbro Torv 1-3, 3rd floor, Aarhus (with regular Friday bar full of geeks ;)
  - www.comiit.com
  - http://www.bigdatadenmark.dk

- Vladimir Smida
  - vladimir@comiit.com
  - https://www.linkedin.com/in/vsmida

- Rasmus Nygaard Andersen
  - rasmus@comiit.com
  - https://www.linkedin.com/in/rasmusnygaardandersen