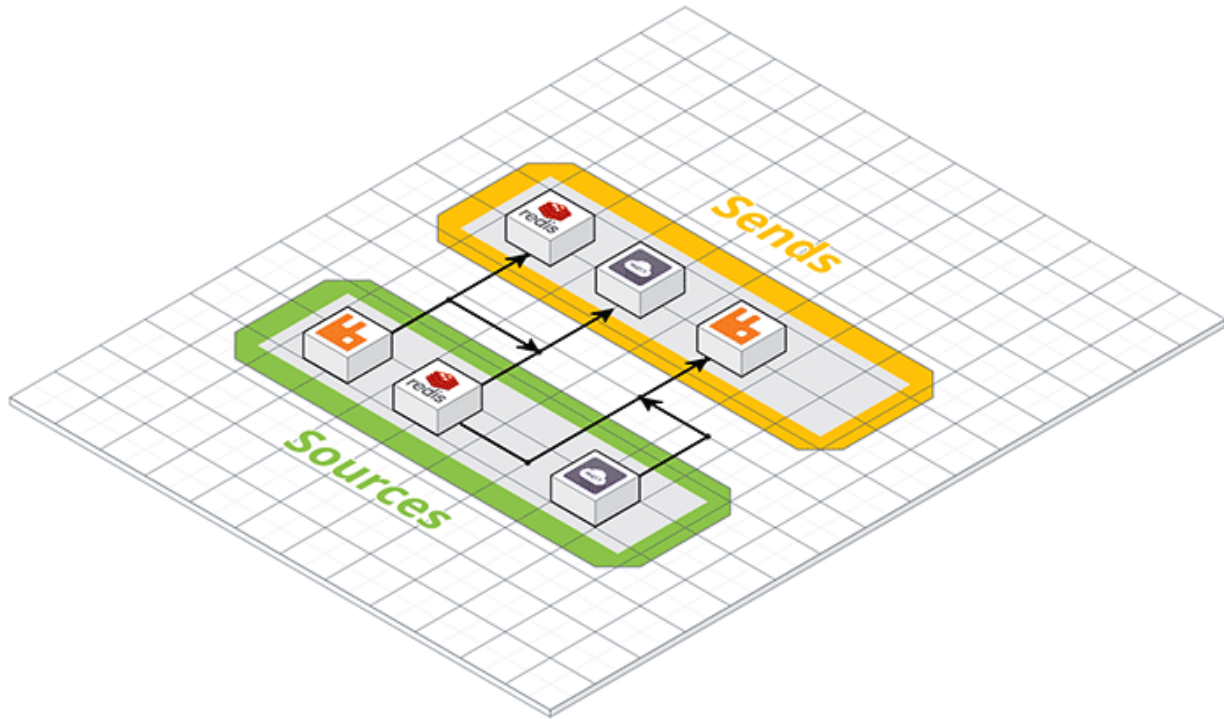

nanopipe Documentation

Release 0.1.0

Christian S. Perone

October 19, 2016

1	Nanopipe features	3
2	Contents	5
2.1	Introduction	5
2.2	Overall Architecture and Concepts	5
2.3	Installation	7
2.3.1	Requirements	7
2.3.2	Building from source	7
2.4	Using the library	7
2.4.1	How does it looks like ?	7
2.4.2	MQTT Source/Send	8
2.4.3	AMQP Source/Send	10
2.4.4	Redis Source/Send	10
2.4.5	C++ Stream Send	10
2.4.6	WebSocket Source/Send	10
2.5	Contributing	10
2.6	License	11
3	Indices and tables	13



Nanopipe is a library that allows you to connect different message queue systems (but not limited to) together. Nanopipe was built to avoid the glue code between different types of communication protocols/channels that is very common nowadays. An example of this is: you have an application that is listening for messages on an AMQP broker (ie. RabbitMQ) but you also have a Redis pub/sub source of messages and also a MQTT source from a weird IoT device you may have. Using Nanopipe, you can connect both MQTT and Redis to RabbitMQ without doing any glue code for that. You can also build any kind of complex connection scheme using Nanopipe. For more information, take a look at the [Overall Architecture and Concepts](#).

Visit the [Nanopipe Github repository](#) for sources, issues, etc. If you want to contribute, take a look at our [Contributing guideline](#).

Nanopipe features

- Open-source (Apache License)
- Written in modern C++
- Valgrind clean
- Declarative definition of the connection graph
- Uses well established event loops (libuv, etc)
- Scalable (sources/sends are threaded)
- Simple producer/consumer queues between source/sends, with very low synchronization overhead
- Multiple source/sends share the same message memory
- Supports AMQP, MQTT, Redis, WebSockets, C++ Streams, etc
- Easy to write new source/sends
- Supports complex architectures

Note: Note that this framework is in active development and it is still in **beta release**. Feel free to contribute !

Contents

2.1 Introduction

Nanopipe is a library that allows you to connect different messaging queue systems (but not limited to) together. Nanopipe was built to avoid the glue code between different types of communication protocols/channels that is very common nowadays.

The development of Nanopipe is still going on and we hope to get help from community to improve it, to implement more sources/sends. There are still many item on our roadmap such as a textual (or yaml, etc) DSL for describing the connection graphs and also a command-line utility to use the library in order to avoid the use of the C++ library itself.

Visit the [Nanopipe Github repository](#) for sources, issues, etc.

2.2 Overall Architecture and Concepts

Nanopipe has two major concepts:

Sources: the sources are producers of messages, they will connect to a source of message and then they will deliver these messages to the **sends**.

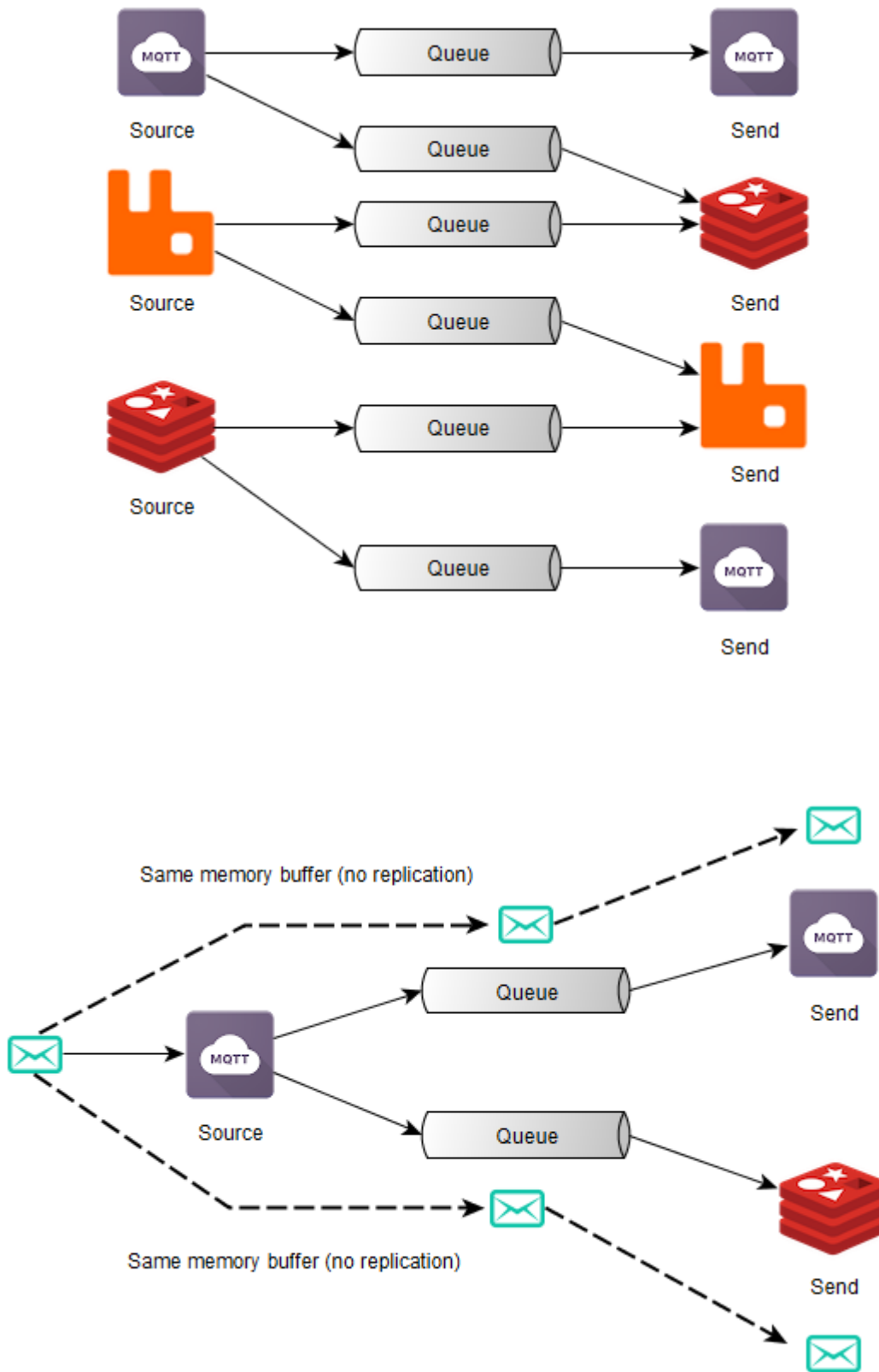
Sends: the sends (or senders) will consume the messages delivered by the sources to another (or even the same) messaging broker.

Note that you can have Sources and Sends that aren't strictly message queues. You can have for instance a *send* that will call a HTTP/HTTPS REST API upon receiving a message from a source.

This architecture is very flexible:

- A source can have multiple sends connected, which means that all the sends will access the same shared memory portion;
- Every source/send will run on its own thread, which means that the receiving of messages on a source will not impact performance on another source or send.

Internally, Nanopipe uses a producer/consumer queue with very low synchronization requirements in order to exchange messages between sources and sends (you can also control the size of this internal message queue). This internal message queue will be instantiated for every source/send connection, so Nanopipe will have a different message queue for each source/send connection, however the message delivered by one source is shared between all its sends.



2.3 Installation

This section describe how to install Nanopipe.

2.3.1 Requirements

To use Nanopipe (and also to compile it), you'll need to install the following requirements:

- libuv \geq 1.9.1 ([official site](#))
- hiredis \geq 0.13.3 ([official site](#))
- AMQP-CPP \geq 2.6.2 ([official site](#))
- Mosquitto \geq 1.4.10 ([official site](#))
- uWebSocket \geq 0.10.12 ([official site](#))

Note: Nanopipe uses some bleeding edge version of some of the libraries above. Some distributions (such as Ubuntu) has packages only for older versions of these libraries, so install them using source.

2.3.2 Building from source

After installing the requirements, clone the Nanopipe repository:

```
git clone https://github.com/perone/nanopipe.git
```

After that, create a new directory called *build* and then use *cmake* to build it:

```
cd nanopipe
mkdir build && cd build
cmake ..
make -j4
```

To install the library in your system, execute:

```
sudo make install
```

2.4 Using the library

This section describes how to use the library and also shows some examples using the C++ API.

Note: Bindings for Python will be developed in near future. If you are interested in contributing, let us know.

2.4.1 How does it looks like ?

Here is an example of an application using the Nanopipe library for you to get a taste on how to declare the sources/sends using different systems.

```
#include <chrono>
#include <nanopipe/nanopipe.hpp>

int main(int argc, char **argv)
{
    nanopipe_init();

    AMQPSource amqp_source1("localhost", 5672, "my-queue");
    AMQPSend amqp_send1("localhost", 5672, "exch", "anykey");
    RedisSource redis_source1("localhost", 6379, "mytopic");
    MQTTSource mqtt_source1("broker.hivemq.com", 1883, "nanopipe");
    RedisSend redis_send1("localhost", 6379, "sink");
    MQTTSend mqtt_send("broker.hivemq.com", 1883, "sinkpipe");
    StreamSend stream_send1(&std::cout);

    redis_source1.addSend(&redis_send1);
    redis_source1.addSend(&mqtt_send);
    redis_source1.addSend(&stream_send1);
    redis_source1.addSend(&amqp_send1);

    mqtt_source1.addSend(&mqtt_send);
    mqtt_source1.addSend(&stream_send1);
    mqtt_source1.addSend(&redis_send1);

    amqp_source1.addSend(&stream_send1);

    NanoManager manager;
    manager.addSource(&amqp_source1);
    manager.addSource(&redis_source1);
    manager.addSource(&mqtt_source1);

    manager.addSend(&stream_send1);
    manager.addSend(&redis_send1);
    manager.addSend(&amqp_send1);
    manager.addSend(&mqtt_send);
    manager.startAll();

    std::this_thread::sleep_for(std::chrono::milliseconds(10000));

    manager.stopAll();
    manager.waitAll();

    return 0;
}
```

This application will execute, wait for 10 seconds and then it will stop all source/send threads.

2.4.2 MQTT Source/Send

The MQTT Source and Send implemented on Nanopipe is based on Mosquitto library. Here is an use case example where we want to get messages from a MQTT broker and then publish them on Redis. You can test this example application using the [HiveMQ MQTT broker](#) where you can use your browser to connect and publish on their broker. You'll also need a Redis server instance running.

```
1 #include <nanopipe/nanopipe.hpp>
2
3 int main(int argc, char **argv)
```

```

4 {
5     nanopipe_init();
6
7     // Instantiate the source and the send
8     MQTTSource mqtt_source("broker.hivemq.com", 1883, "wstopic");
9     RedisSend redis_send("localhost", 6379, "wstopic");
10
11    // Connect the send into the source
12    mqtt_source.addSend(&redis_send);
13
14    // Instantiate the manager
15    NanoManager manager;
16    manager.addSource(&mqtt_source);
17    manager.addSend(&redis_send);
18
19    // Wait forever
20    manager.startAll();
21    manager.waitAll();
22
23    return 0;
24 }

```

As you can see, the example is very simple and intuitive. We first instantiate the source and the send and then we just connect both before starting our manager, that will be responsible for executing the graph.

Every message published on the topic **wstopic** on the MQTT broker will be sent to the Redis topic **wstopic**. Everything without struggling to write glue code.

You can also use the MQTT Send to send messages to a MQTT broker instead of using it as a source, here is an example of using it on a problem where you want to send messages arriving on a RabbitMQ (AMQP) queue to a MQTT broker:

```

1  #include <nanopipe/nanopipe.hpp>
2
3  int main(int argc, char **argv)
4  {
5      nanopipe_init();
6
7      // Instantiate the source and the send
8      AMQPSource amqp_source("localhost", 5672, "myqueue");
9      MQTTSend mqtt_send("broker.hivemq.com", 1883, "mytopic");
10
11     // Setting the MQTT QoS to 2
12     mqtt_send.setQos(2);
13
14     // Connect the send into the source
15     amqp_source.addSend(&mqtt_send);
16
17     // Instantiate the manager
18     NanoManager manager;
19     manager.addSource(&amqp_source);
20     manager.addSend(&mqtt_send);
21
22     // Wait forever
23     manager.startAll();
24     manager.waitAll();
25
26     return 0;
27 }

```

2.4.3 AMQP Source/Send

TODO

2.4.4 Redis Source/Send

TODO

2.4.5 C++ Stream Send

TODO

2.4.6 WebSocket Source/Send

2.5 Contributing

If you want to contribute, first of all, take a look at our Github repository to check if the feature you will work isn't already being developed. If not, create an issue on Github to discuss what you're going to work on and if it is a good idea. Make sure to describe what and how you want to implement a feature or contribution.

Once you have discussed what you want to work on, then you will want to:

- Fork the official repository.
- Clone your fork:

```
git clone git@github.com:<your-username>/nanopipe.git
```

- Make sure tests are passing for you:

```
make && make test
```

- Create a topic branch:

```
git checkout -b new-feature
```

- Add tests and code for your changes.
- Once you're done, make sure all tests still pass:

```
make && make test
```

- Commit and push to your fork.
- Create an issue with a link to your patch.
- Sit back and enjoy.

There are other ways to help:

- Fix a bug or share your experience on issues
- Improve the documentation
- Help maintain or create new client libraries
- Improve this very website

2.6 License

Copyright 2016 Christian S. Perone

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Indices and tables

- `genindex`
- `modindex`
- `search`