Global Sensors Networks: ØMQ (zeromq) experiment

Introduction

The GSN server has been designed to work in a distributed fashion, having multiple instances that are able to communicate together. However the internal communication has been implemented on top of an HTTP and XML based protocol. Two versions of this protocol exists. The first one is push based and works in a publish-subscribe manner, whereas the second one is pull-based and can work even if the "client" GSN is behind a NAT (doesn't have a public IP address). The main advantage of this protocol is its ease of debug and implementation on other tools (as long as an xml and http library is provided), but the overhead for processing the data distribution is not negligible.

An alternative has been implemented, using ØMQ[1] and the java serialization library Kryo[2]. The principle is similar to the push version of the HTTP wrapper, using the PUB and SUB sockets, as shown on Figure 1. To allow the simultaneous use of internal (using the "inproc" sockets) and external connections (by IP address) a proxy is taking care of forwarding the subscriptions and the data. It also serve as a directory, listing the available sensors and their data structure for external connections.

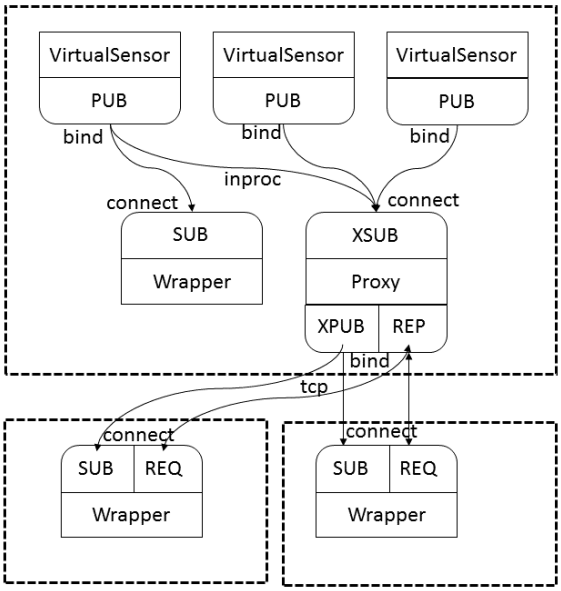

FIGURE 1. BLOCK DIAGRAM OF THE ØMQ COMMUNICATION

In the following, we evaluate the performances of the Ømq data distribution process and compare it with the push remote wrapper. We also show the difference in using Kryo serialization instead of the xml serialization.

[1] See http://zeromq.org/intro:read-the-manual for a detailed tutorial on ØMQ
[2] https://github.com/EsotericSoftware/kryo

Test bed

To evaluate the inter-server communication, we imagined a use-case where each server is receiving or generating a stream of data and need to share it with all the other servers, in a kind of worst case scenario.

We deployed 24 GSN servers, each on its own virtual machine, in Proxmox containers, distributed over 9 physical machines (3, 3, 3, 3, 3, 3, 2, 2,2). The virtual machines are provisioned with 2 cores (2.66GHz) and 3GB of RAM. Each GSN server has 25 virtual sensors: one generating data every 10 ms and 24 connected to the other instances (including itself). The storage was kept in memory using the H2:mem database to reduce the disk writing overload. The number of connections in the pool was set to 50 to ensure there will be enough connection available for every virtual sensors and wrappers. (The debugging output showed that actually only half of them were used, meaning one per virtual sensor)

In the first experiment, the remote wrapper was used to connect the 24 virtual sensors, in the second one the zeromq wrapper was used with xml serialization and finally in the last experiment zeromq was using the Kryo serialization. Each experiment lasted around 20 minutes during which two snapshots were taken. One after a few minutes to ensure that all connection have been established and one 15 minutes after the first one. These snapshots contained the number of element processed by each virtual sensors, counted using the CounterVS virtual sensor.

After running all experiments we noticed that Proxmox seems to limit the bandwidth of its virtual machines outgoing traffic, and even when using the smallest stream element as possible with only one field, it still was the bottleneck in some experiments. Further experiments may be conducted when we find the root of this problem.

Results

In the first execution of the test-bed, using the remote wrapper and xml serialization, the CPU load of the virtual machines stayed around 36% and the network traffic was around 120kbps (Figure 2.). The counter on the virtual sensor generating data indicated a rate of 90 element per seconds. This means GSN needs 1 ms to generate the element and then waits for 10 ms before generating the next one. The network traffic is perfectly symmetric as every server is sending and receiving to, respectively from, all the others.

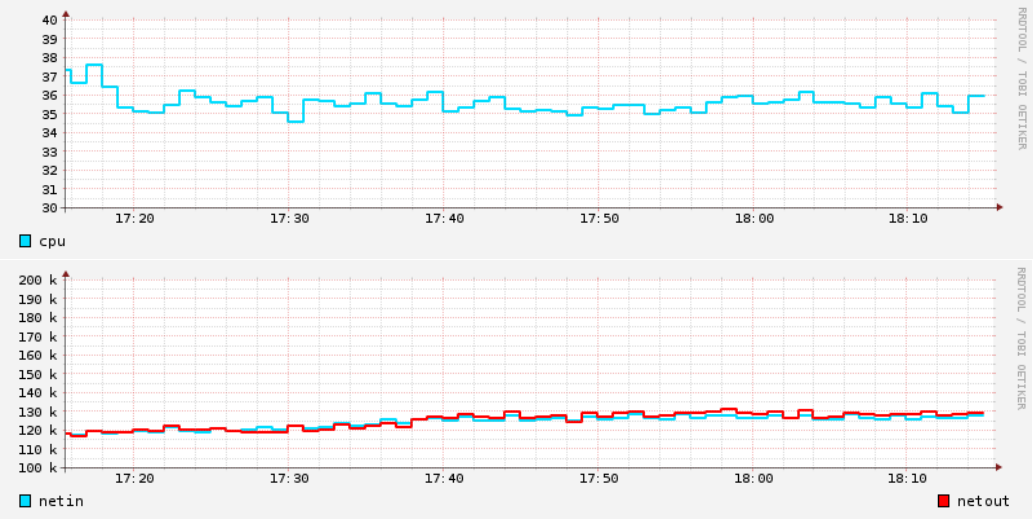

FIGURE 2. CPU AND NETWORK LOAD WHEN USING THE REMOTE WRAPPER.

However, with an average of 7 elements per second processed per virtual sensor connected with the remote wrapper, it is clear that it cannot follow the element production. Here the network is the bottleneck.

The second and third run, using the zeromq wrappers had pretty similar behavior regarding the CPU load and network traffic. CPU was almost at its maximum and network showed some differences in incoming versus outgoing traffic (Figure 3.). This can be explained by looking at the distribution of the generation rate among the servers, on Figure 4. All the servers received the same amount of data (same incoming traffic), but the ones generating less elements had also less data to send (lower outgoing traffic).

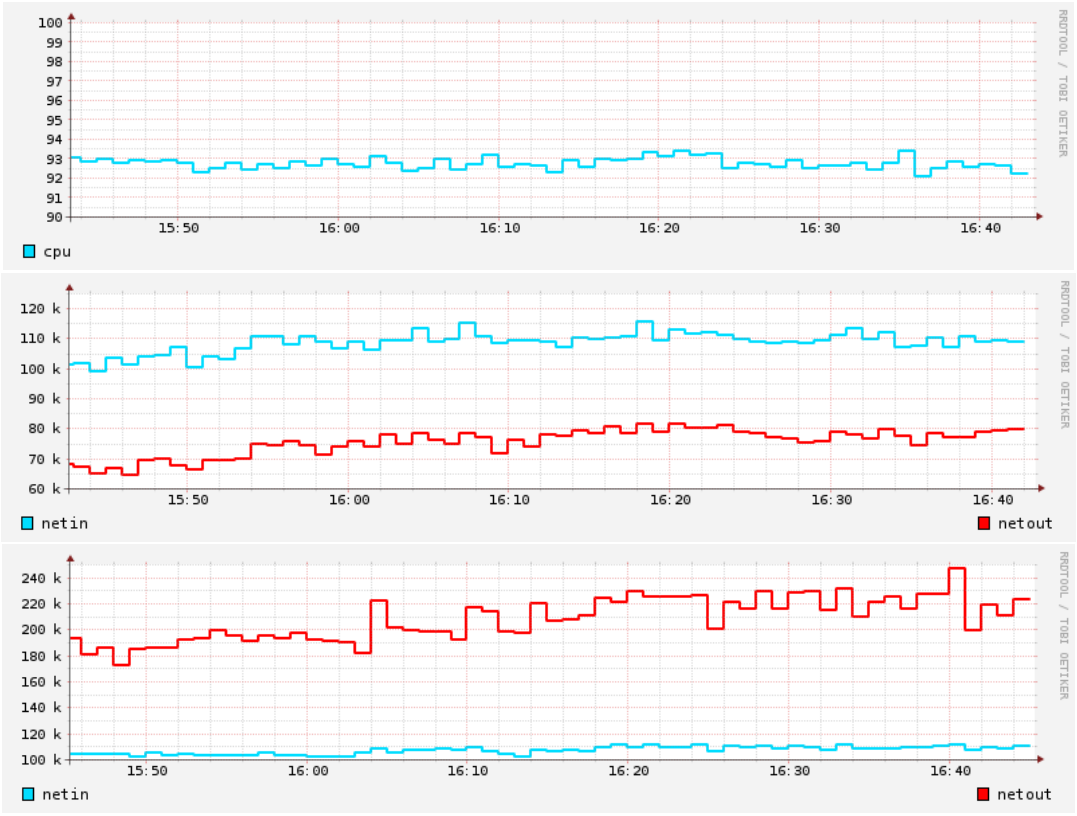

FIGURE 3. CPU AND NETWORK LOAD WHEN USING ZEROMQ WRAPPER. THE NETWORK LOAD SHOWED SOME VARIATION AMONG THE MACHINES.

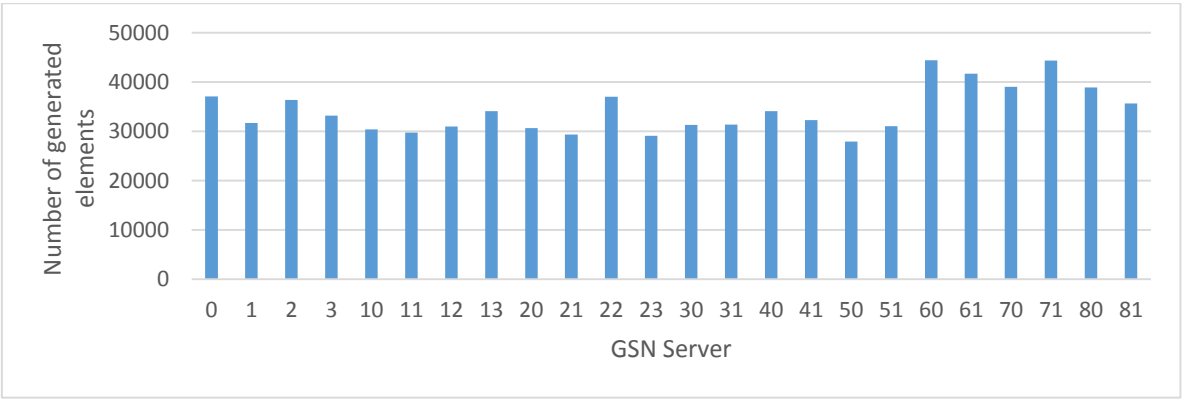

FIGURE 4. DISTRIBUTION OF THE NUMBER OF GENERATED ELEMENT ON EACH SERVER, USING THE ZEROMQ WRAPPER AND KRYO SERIALIZATION.

In the experiment using xml serialization, the communication protocol being lighter than http, it was possible to send and process twice as much elements per seconds per virtual sensor, 13. But for processing those elements, the CPU was also more solicited (almost 100%) and was not able to keep the production rate which dropped to 57 elements per seconds (Figure 5.).

Finally using the Kryo serialization, the network was saturated with 36 elements per second per virtual sensor. And similarly to the previous experiment the CPU had less time producing elements, around 38 per seconds. In this last experiment we almost reached the maximum performance possible with our virtual machines limitation: 860 elements sent, received and processed per second.

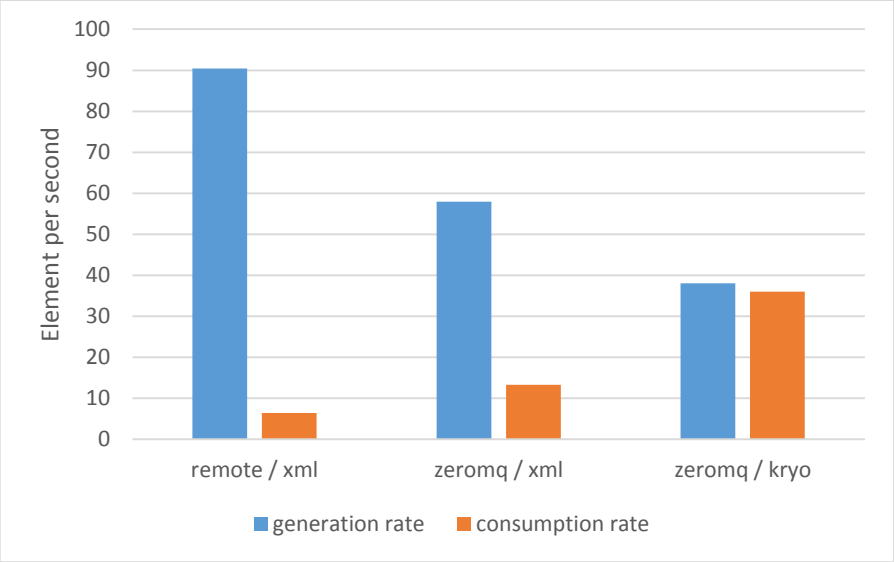

FIGURE 5. SUMMARY OF THE PERFORMANCES OF THE DIFFERENT INTER-GSN COMMUNICATION PROTOCOLS.

Conclusion

These three experiments show that with limited resources the choice of the communication protocol is very important. But, what we gained in performance, we lost it in reliability. The current implementation of the zeromq wrapper doesn't offer a way to acknowledge when an element has arrived and even if it is implemented on TCP/IP, it doesn't guarantee that no element is lost (when the server is down or restarting).

The choice of the zeromq wrapper for communication should be made with this tradeoff in mind.