A Performance Analysis Plugin for DAQPIPE

CERN Summer Student Program 2015 – Project Report

Kilian Lieret

Supervisors:
Sébastien Valat, Daniel Hugo Campora Perez, Dr. Niko Neufeld

February 11, 2016

Contents

1 Project Overview

In 2020 the Data Acquisition (DAQ) of the LHCb experiment will be updated to feature a trigger-free readout. This requires an event builder network consisting of about 500 nodes with a total network capacity of 4 TBytes/s[1]. `DAQPIPE` (Data Acquisition Protocol Independent Performance Evaluator) is a tool to simulate and evaluate the performance of such a DAQ system.
The current implementation of `DAQPIPE` only gives rough feedback about the event building rate. The aim of this 10-week summer student project was to implement network monitoring for a more detailed performance evaluation of different transport protocols and to spot potential bottlenecks.

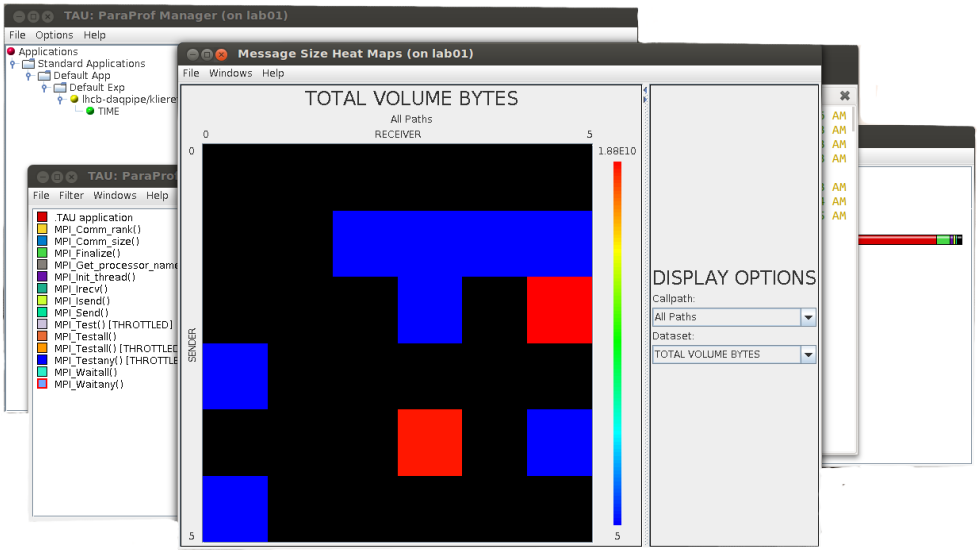

Figure 1: Communication matrix with `Tau` and `ParaProf`.

2 DAQ

The DAQ system as simulated by `DAQPIPE` mainly consists of a network of *Readout Units* (RUs) and *Builder Units* (BUs). The RUs collect incoming data fragments from different subdetectors. All of the data fragments from one event are then sent to one of the BUs, which combines all data fragments into one data package ("Event Building"). The receiving BU is selected by the *Event Manager* (EM) which tries to achieve a balanced load distribution among the BUs. The EM is subdivided into the *Event Manager Listener* (EML) and the *Event Manager Consumer* (EMC).

3 Existing Tools

The first stage of the project was to get familiar with `DAQPIPE` and the concept of performance monitoring. To that end, `DAQPIPE` was run together with `Tau`[1] to obtain performance data which was then plotted with `ParaProf`[2], `JumpShot`[3] and `Vampir`[4].

4 A Custom Made Performance Analysis Tool

In the second stage of the project, a light-weight performance analysis tool was written from scratch.

[1] http://www.cs.uoregon.edu/research/tau/home.php
[2] https://www.cs.uoregon.edu/research/tau/docs/paraprof/
[3] http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/
[4] https://www.vampir.eu/

4.1 General Idea

As an elegant way to keep track of the communication between the different nodes without having to make any alterations to the DAQPIPE sourcecode itself, the basic idea was to wrap around the methods of the underlying C++ communication library in order to inject additional code that counts the number of transferred messages and bytes. To that end, a dynamic shared library was created, redefining all relevant library methods. By setting the LD_PRELOAD environment variable to the path of the modified library it was then ensured that it is loaded before the original library, thereby overriding the original methods.

Among the different communication protocols, the MPI library was selected for this feasability study.

4.2 Implementation

On each node, the modified MPI library initializes a Counter object. In addition to that, a Logger object is initialized on the node of the Event Manager.

- The Counter holds internal variables that store the total number and size of the messages that are sent from this node to another. Each MPI communication method that *sends* data is modified to update these variables. After COUNTER_TIMEOUT μs the Counter sends the collected data to the Logger node and resets its variables. Since this timeout criterion is only checked when one of the wrapped MPI methods[5] is called, the actual timedifferences $\Delta t_{\texttt{Cnt}}^{(\#\text{node})}$ will vary (and always be larger than COUNTER_TIMEOUT), depending on how often the wrapped MPI methods are called. However, the actual timedifferences are sent to the Logger as well, allowing for a correct calculation of the data rates afterwards.
- The Logger works similar: it has analogue internal variables (one for each Counter object) that get updated once the Logger receives data from a Counter. If the Logger receives more than one dataset from a Counter, the corresponding values are added. In particular the timedeltas $\Delta t_{\texttt{Log}}^{(\#\text{node})}$ that are saved to the file are the sum of the $\Delta t_{\texttt{Cnt}}^{(\#\text{node})}$ of all received datsets.[6] Once both a timeout criterion is reached (ideally once every LOGGER_TIMEOUT μs) and the Logger received at least one dataset from each Counter, the collected data gets appended to a file and the variables are reset.

Note that the slightly varying timedifferences corresponding to which the byterates are calculated do not impair the validity of the of the calculated byterates for the corresponding

[5] right now the wrapped methods are the synchronous and asynchronous send and receive methods, the wait and the test methods, as well as the initialisation method

[6] to keep $\Delta t_{\texttt{Log}}^{(\#\text{node})}$ and $\Delta t_{\texttt{Cnt}}^{(\#\text{node})}$ distinct, I call the former *timedeltas* and the latter *timedifferences*

timeframes, though of course the time precision suffers.[7]

4.3 Visualisation

A small monitor was written in `Python` with the `PyQt` library, parsing plain text output files and showing the total bandwidth transmitted as well as the distribution of the message size (Fig. 2).

The plots of this report are generated by `Python` with the `PyPlot` library.

4.4 Exemplaric Results

The data of all shown plots stems from consecutive DAQPIPE runs with similar parameters, allowing for some rough cross-checks. All tests were run with `COUNTER_TIMEOUT` = `LOGGER_TIMEOUT`.

- The easiest obtainable (since time-resolution independent) results are communication matrices as shown in Figure 3.
- Plots of the timedeltas $\Delta t_{\texttt{Log}}^{(\#\text{node})}$ vs. the time are shown in Fig. 4 for `COUNTER_TIMEOUT`s of 50 resp. 200ms. Though the average timedelta variation $\Delta t_{\texttt{Log}}^{(\#\text{node})}/\texttt{COUNTER\_TIMEOUT}$ (and thus $\Delta t_{\texttt{Cnt}}^{(\#\text{node})}/\texttt{COUNTER\_TIMEOUT}$) is rather low, the graphs show sharp peaks with an amplitude of roughly 2 that seem to appear more or less periodically. These "2-peaks" can be explained as follows: With the current setup, the time deviations $\Delta t_{\texttt{Cnt}}^{(\#\text{node})} - \texttt{COUNTER\_TIMEOUT}$, though being small, accumulate, causing the different nodes to get out of sync more and more. At some point, a situation like the one depicted in Fig. 5 will occur and the logger will receive more than one dataset from the same node before writing out, causing all data values (including $\Delta t_{\texttt{Log}}^{(\#\text{node})}$) to be roughly twice of their normal value.[8] An overview over the time deviations for the probed `COUNTER_TIMEOUT`s is given in Fig. 6.
- The time dependency of the data rates for the two RUs, measured with `COUNTER_TIMEOUT` = 50, 200ms and 1000ms is shown in Fig. 7 and Fig. 8.
- Finally, Fig. 9 shows the sum of all the data rates and the number of sent messages vs. time.

[7] However, for graphs showing the total byterate of *several* nodes, the byterates need to be redistributed to constant timedifferences. Given a set of rates r_i calculated for timeframes $[t_i, t_{i+1}]$, the values corresponding to timeframes $[\bar{t}_j, \bar{t}_{j+1}]$ can be approximated by:

$$\bar{r}_j \;=\; \frac{1}{\left|[\bar{t}_j, \bar{t}_{j+1}]\right|} \cdot \sum_i \Big|[\bar{t}_j, \bar{t}_{j+1}] \cap [t_i, t_{i+1}]\Big| \cdot r_i. \tag{1}$$

[8] Assuming that one node runs almost precise ($\Delta t_{\texttt{Cnt}}^{(1)} \approx 0$s) and a second node shows average timedifferences $\left\langle \Delta t_{\texttt{Cnt}}^{(2)} \right\rangle$, the distance of two 2-peaks in the $\Delta t_{\texttt{Log}}^{(1)}$ distribution should be about $\texttt{COUNTER\_TIMEOUT}^2 / \left\langle \Delta t_{\texttt{Cnt}}^{(2)} \right\rangle$. As $\left\langle \Delta t_{\texttt{Cnt}}^{(2)} \right\rangle$ decreases with `COUNTER_TIMEOUT`, the distance grows at least with $\texttt{COUNTER\_TIMEOUT}^2$.

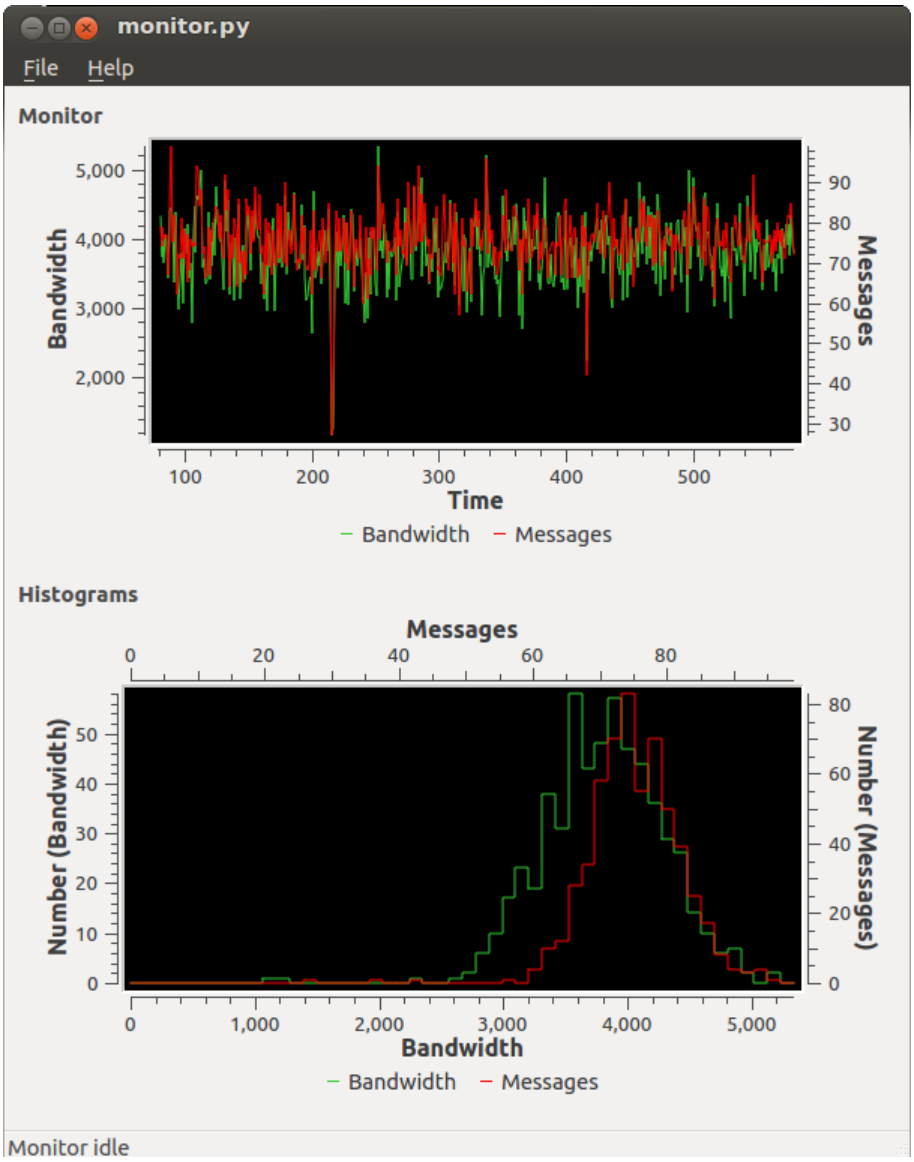

Figure 2: Simple live monitor written in `Python` with the `PyQt` library, continuely parsing data that gets appended to a plain text file (picture shows mock data).

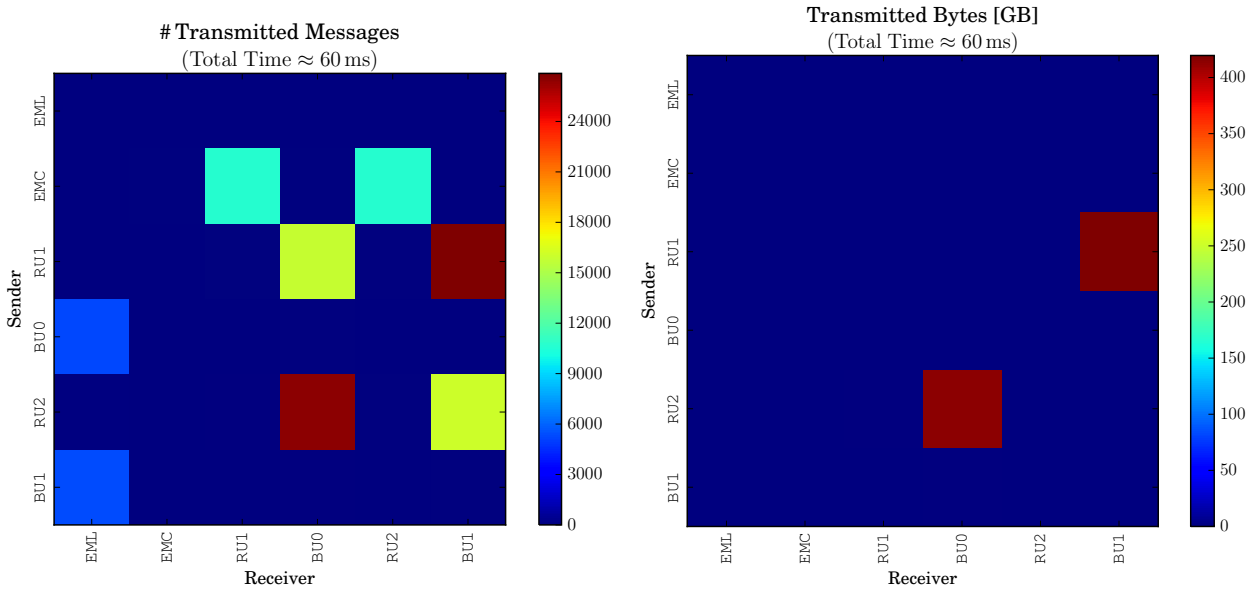

Figure 3: Total sum of the number of transmitted messages and transmitted data, split up by sender and receiver.

4.5 Usage

Once the shared library is compiled, running `DAQPIPE` together with the monitoring is as easy as setting the `LD_PRELOAD` variable to the path of the `.so` file. Due to some pecularities with `mpirun` and environmental variables, a (trivial) wrapping script `run_wrap.sh` has to be used to do this. Given a normal `DAQPIPE` call, simply modify the call by preceding the `DAQPIPE` executable (`eb_app`) with the call to this script, e.g.:

```
mpirun --map-by ppr:2:node -mca btl openib,self\
   -mca btl_openib_warn_default_gid_prefix 0\
   -machinefile config/labs_02_04.labs\
   logging/run_wrap.sh build/eb_app -f 100000 -s 10240 -c 2 -t 24
```

For more information, please see the readme files.

4.6 Improvements

As two-peaks and similar precision issues are connected to the nodes getting out of sync over time, modifications to the timing of the Counter send outs could lead to improvements with the time precision, e.g.:

- Instead of a fixed `COUNTER_TIMEOUT`, the `Counter` keeps track of whether it is ahead of behind the schedule and adjusts the `COUNTER_TIMEOUT` correspondingly.
- All nodes periodically call a method like `MPI_BARRIER` which waits till all nodes have called this function, thereby enforcing synchronisation among the nodes.

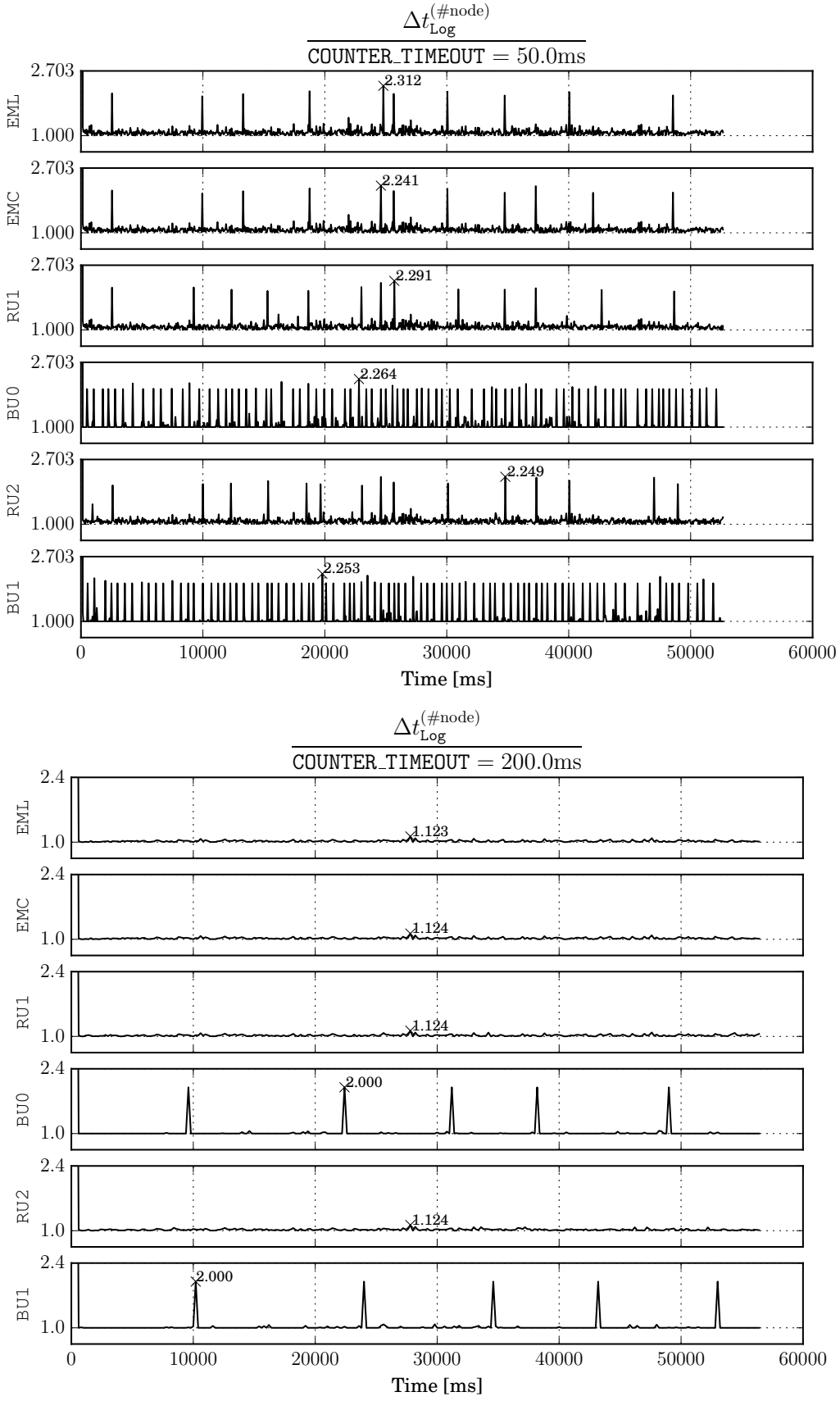

Figure 4: Timedelta variations for COUNTER_TIMEOUT = 1s and COUNTER_TIMEOUT = 200ms. The "×" marks highlight the maxima of the shown graphs for $t > 5$s.

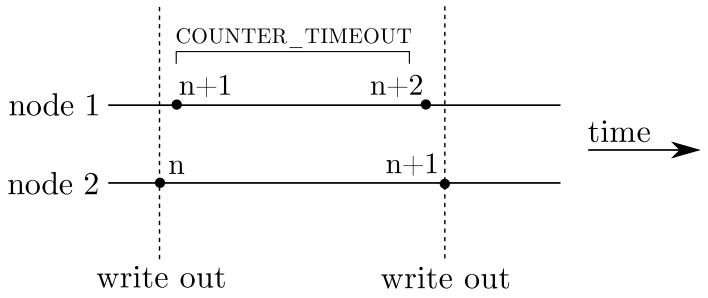

Figure 5: Examplaric scenario for 2-peaks. Black dots indicate that the corresponding counter sends data to the `Logger`, the close-by number indicates the number of the send out. Suppose that node 2 shows a higher average $\Delta t_{\texttt{Cnt}}$ than node 1, causing it to more and more fall behind of node 2, until it is more than one `COUNTER_TIMEOUT` behind. As the `Logger` only writes out the data once it has received data from all `Counter`s, the `Logger` will always "wait" for node 2. In the depicted scenario this makes it possible that the `Logger` receives two datasets from node 1, which then show up as a 2-peak. Afterwards both nodes are (from the standpoint of the `Logger`) almost synchronous again and the cycle repeats, explaining the periodic nature of the 2-peaks.

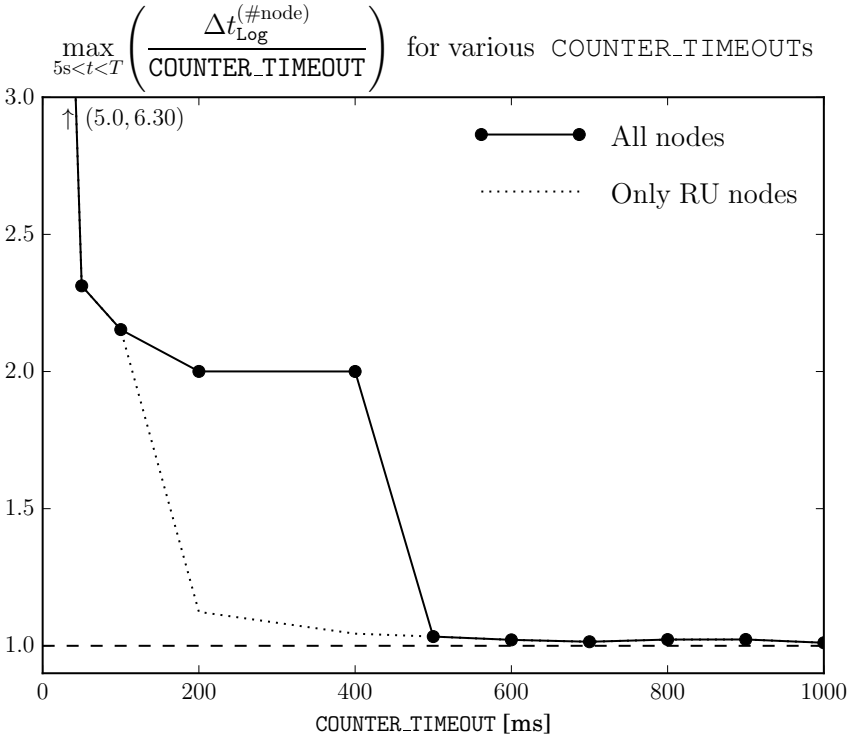

Figure 6: Maximal timedelta variation of for different `COUNTER_TIMEOUT`s. Note that the sampling time of $\approx$ 30s resp. $\approx$ 60s was probably too small for 2-peaks to show up for larger `COUNTER_TIMEOUT`s.

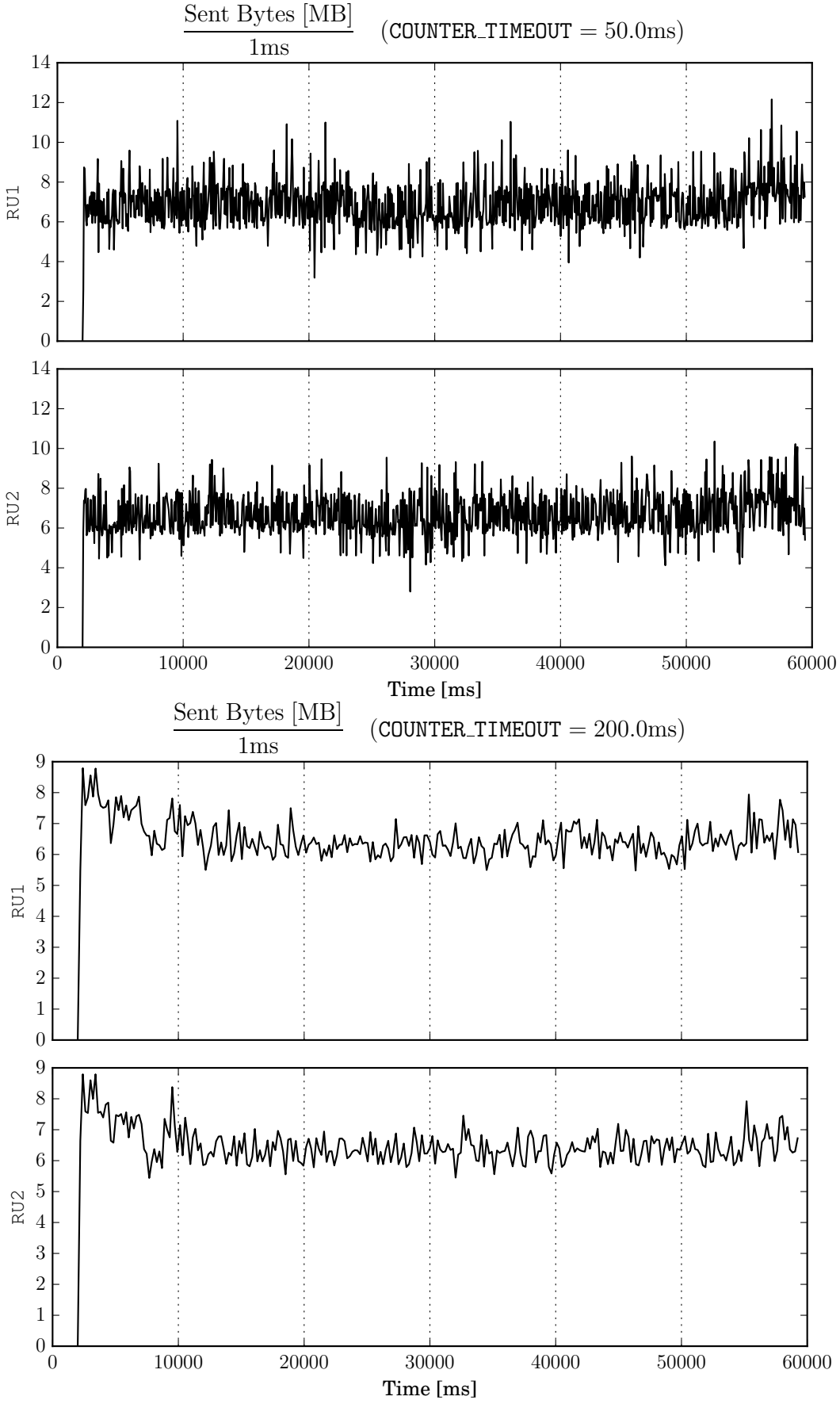

Figure 7: Data rates for the two RUs, measured with a COUNTER_TIMEOUT of 50ms resp. 200ms.

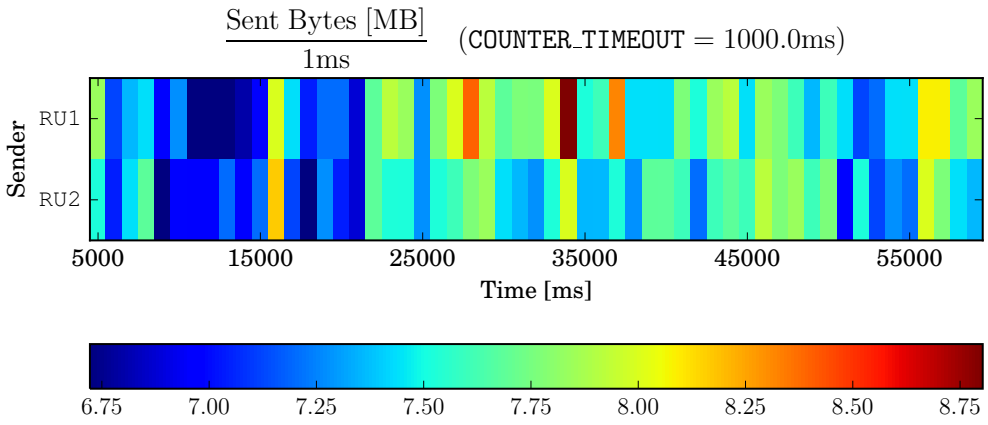

Figure 8: Different visualisation of the data rates per RU.

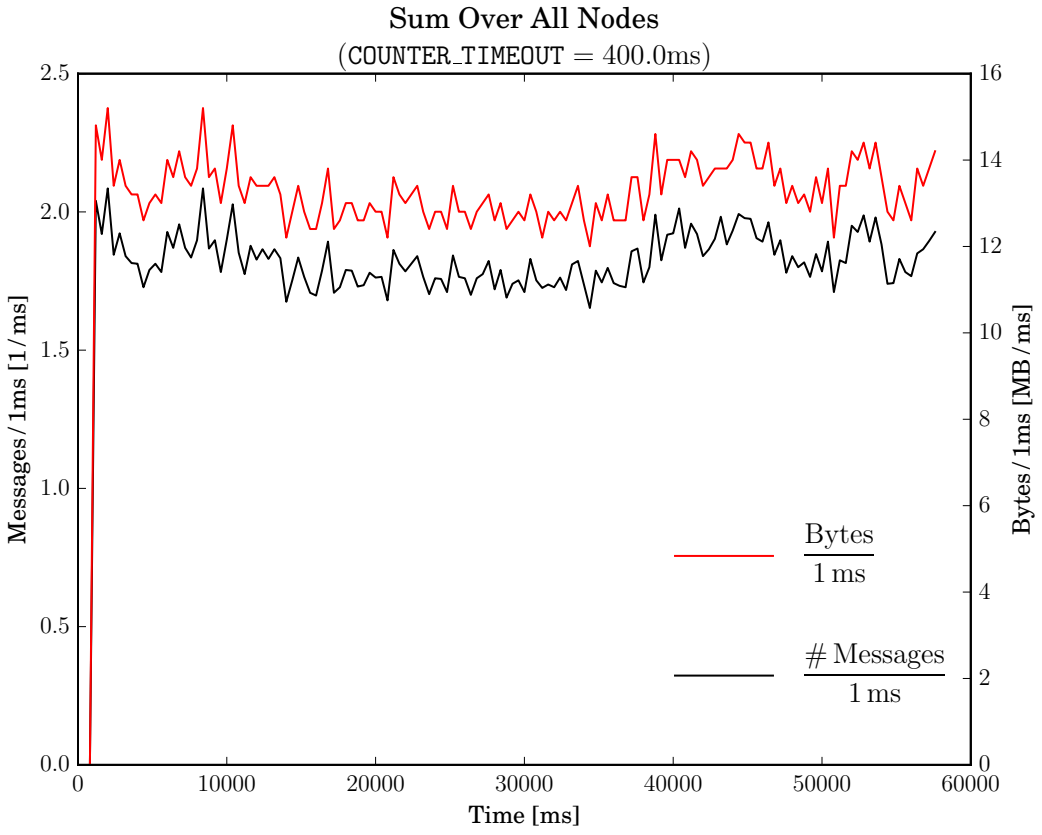

Figure 9: Total rates among all nodes.

5 Conclusion

It was shown that it is possible to create lightweight performance analysis plugins for `DAQPIPE` with relatively low effort. The implementation described in this report is dependent on the used communication protocol/library. Though only the `MPI` library was tested, the code should be easily adaptable for the other communication protocols, as the methods used by the underlying library exist in every of these libraries, so that only these method calls have to be replaced.

References

[1] Campana, Pierluigi; Lindner, Rolf, *LHCb Trigger and Online Upgrade Technical Design Report* (Technical Design Report LHCb), CERN-LHCC-2014-016, LHCB-TDR-016 https://cds.cern.ch/record/1701361?ln=en