Summer Student Programme 2016

Candidate
Mattia Cadeddu IT/CS-CE

Supervisors
Stefan Nicolae Stancu
Adam Lukasz Krajewski

NETBENCH

AUTOMATED NETWORK PERFORMANCE TESTING

1. INTRODUCTION

In order to evaluate the operation of high performance routers, CERN has developed the *NetBench* software to run benchmarking tests by injecting various traffic patterns and observing the network devices behaviour in real-time. The tool features a modular design with a Python based console used to inject traffic and collect the results in a database, and a web user interface for visualizing the results.

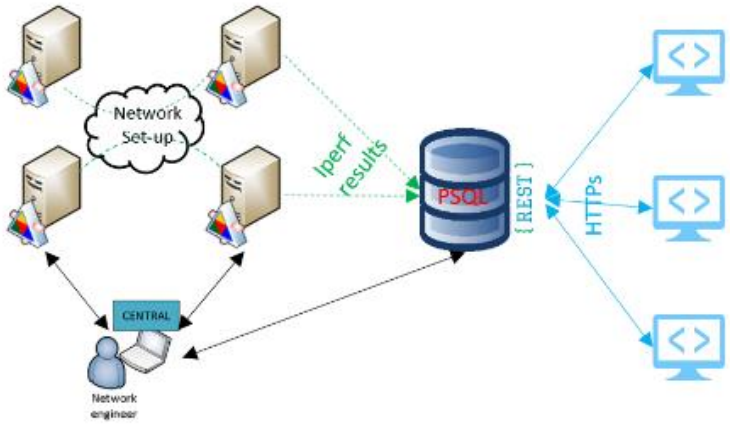


Figure 1 NetBench architecture

Figure 1 depicts the architecture of NetBench. A group of nodes (A) exchanges data, through a network setup, with another group of nodes (B) using several traffic distributions such as [1].

The network setup can be a single Device Under Test (DUT) or a collection of such devices (routers and/or switches).

In order to generate traffic, an agent is deployed and installed in each node. An agent is a Python module that can put traffic into the network using several engines such as [2].

The agents collect statistics (e.g. bandwidth) for all the flows that traverse the network and push them into a PostgreSQL database, which then serves for producing statistics plots.

Agents are controlled by the Central module using the RPC protocol. The central module is also written in Python and it handles the lifecycle of a test, i.e. start/stop the traffic and memorizing the test status (ongoing, finished) into the database.

High-level statistics graphs are provided in a Web UI that has been developed using standard web technologies:

- HTML and CSS to define the structure of the pages,
- Javascript JQuery and D3.js to make the UI interactive and plot the high-level statistics

All the information is retrieved from the database using a Python REST API.

2. RUNNING A TEST

Before starting a test, it is necessary to provide a JSON configuration (see Code 1 below for a sample file) file containing the following information:

- the list of servers in each node group;
- various traffic options such as engine, base-port, topology, type of connection, number of connections between each pair of nodes and possibly some engine options;
- the interfaces to be used for sending data
- the agents' deployment path, the agent name and port number.

```
{"nodes_a": [<nodes name>],
"nodes_b": [<nodes name>],
"traffic": {
        "engine": "<bernd|iperf|iperf3>", "baseport": <number>,
        "topology": "<pairs|partial-  mesh|full-mesh>", "connection": "<undirect|bidirect>",
        "connection-count": <number>, "engine-opts": {…}
},
"logging": "true",
"provisioning": {
        "interfaces": {"nodes_a": [<interface>], "nodes_b": [<interface>]},
        "deployment": {"agent-path": "<path>"}
},
"agent": {"path": "<path>","port": <number>}}
```

Code 1 . Configuration file structure

3. PERSONAL CONTRIBUTIONS

The NetBench project is currently under. My personal contribution can be found in several layers of the tool, from core modules (e.g. agents) to data visualization.

Agent Module. One of the agent jobs is to generate and put traffic into the network using a component called *engine*. On top of the already supported engines (a custom program and *iPerf2*) I have developed an adapter for *iPerf3* (the newest version of iperf) and thus addressed some problems present with the older version (e.g. poor scalability of the number of connections due to hitting a limit on open file descriptors). To summarise, now NetBench can use the latest version of iperf and it can set-up a much larger number of TCP connections than before.

Data Visualization. The data that agents have collected into the database are used to produce high-level statistics reflecting the performance of the DUT, which are exposed via a REST API and used by a web client to generate plots. Before my contribution, the plots generation was too slow for an acceptable user interface experience. By improving code, optimizing the DB queries and adding DB indexes, I have obtained a ~10 fold performance increase, achieving the performance summarized in Table 1.

Nodes	Topology	Connection count	Total no. of connections	Plot visualization delay
20	full-mesh	4	1520	2 – 5 sec.

Table 1 Plot visualization performance

I have changed or added the following visualization functionality:

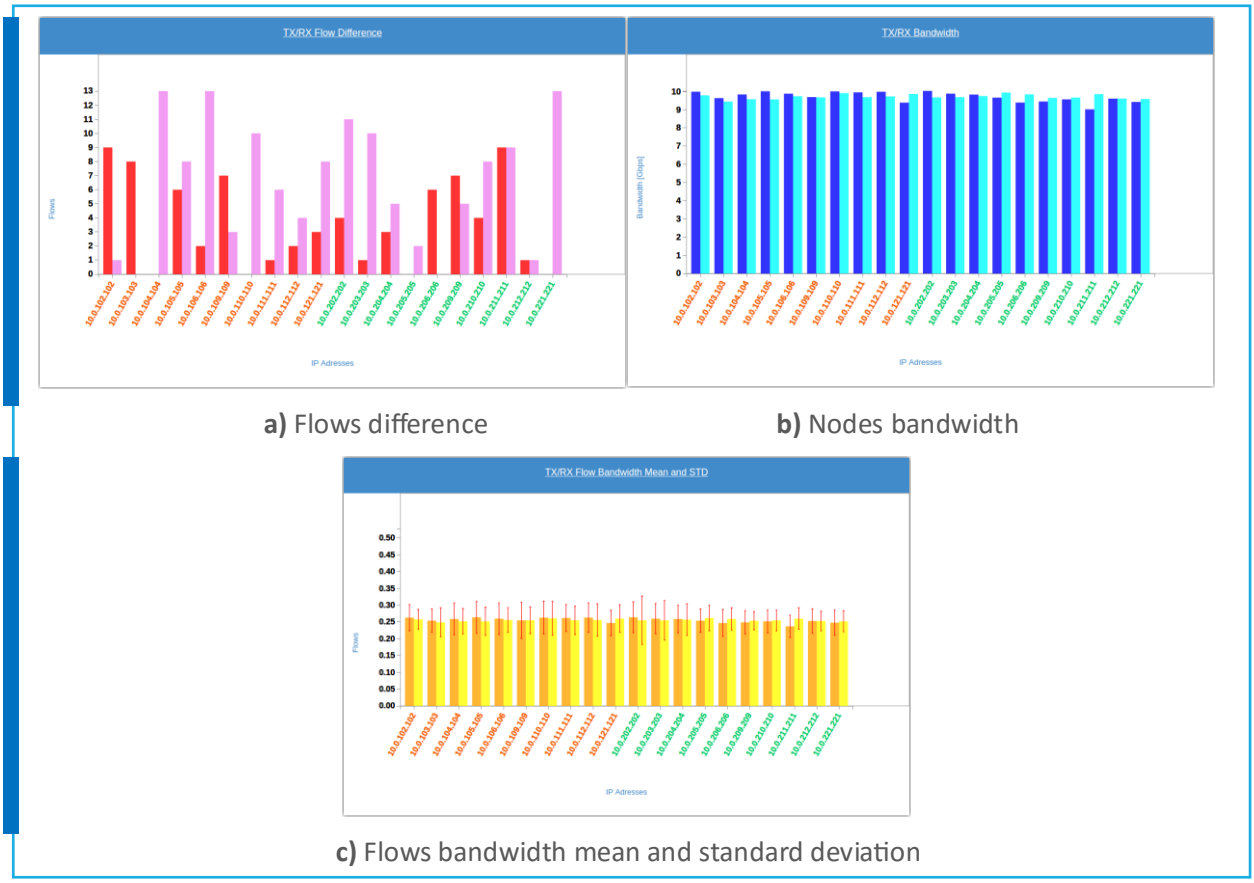
- Added outgoing/incoming bandwidth bar-chart
- Merged outgoing and incoming Flow difference bar-chart
- Merged outgoing and incoming bandwidth (mean and standard deviation) bar-charts
- Added per-pair cumulative throughput heat-map
- Added per-pair flows relative standard deviation heat-map

In order to standardize the appearance of the graphs I have developed a Javascript-JQuery-D3.js library that provides drawing methods which can be used for generating plots with similar look and feel.

PER NODE PLOTS

Figure 2 illustrates the per node plots. Each bar-chart has been created using a dark color bar to indicate outgong traffic and a light color to indicate the incoming traffic of a node. The following plots are produced:

a) **Flow Difference** – shows difference between the expected flows and the seen flows. The expected flows are derived from the configuration file whereas the seen flows are retrieved from the database.
b) **Nodes bandwidth** – shows the outgoing/incoming bandiwdth of the nodes. For each node the bar value is computed summing all the flows bandwidth of the selected node.
c) **Flows bandwidth mean and standard deviation** – shows what is the flows behavior of each node. The bar value represents the mean of all the flows and in order to have an overview of the bandwidth sharing between flows the standard deviation is plotted in red over each bar.



a) Flows difference **b)** Nodes bandwidth

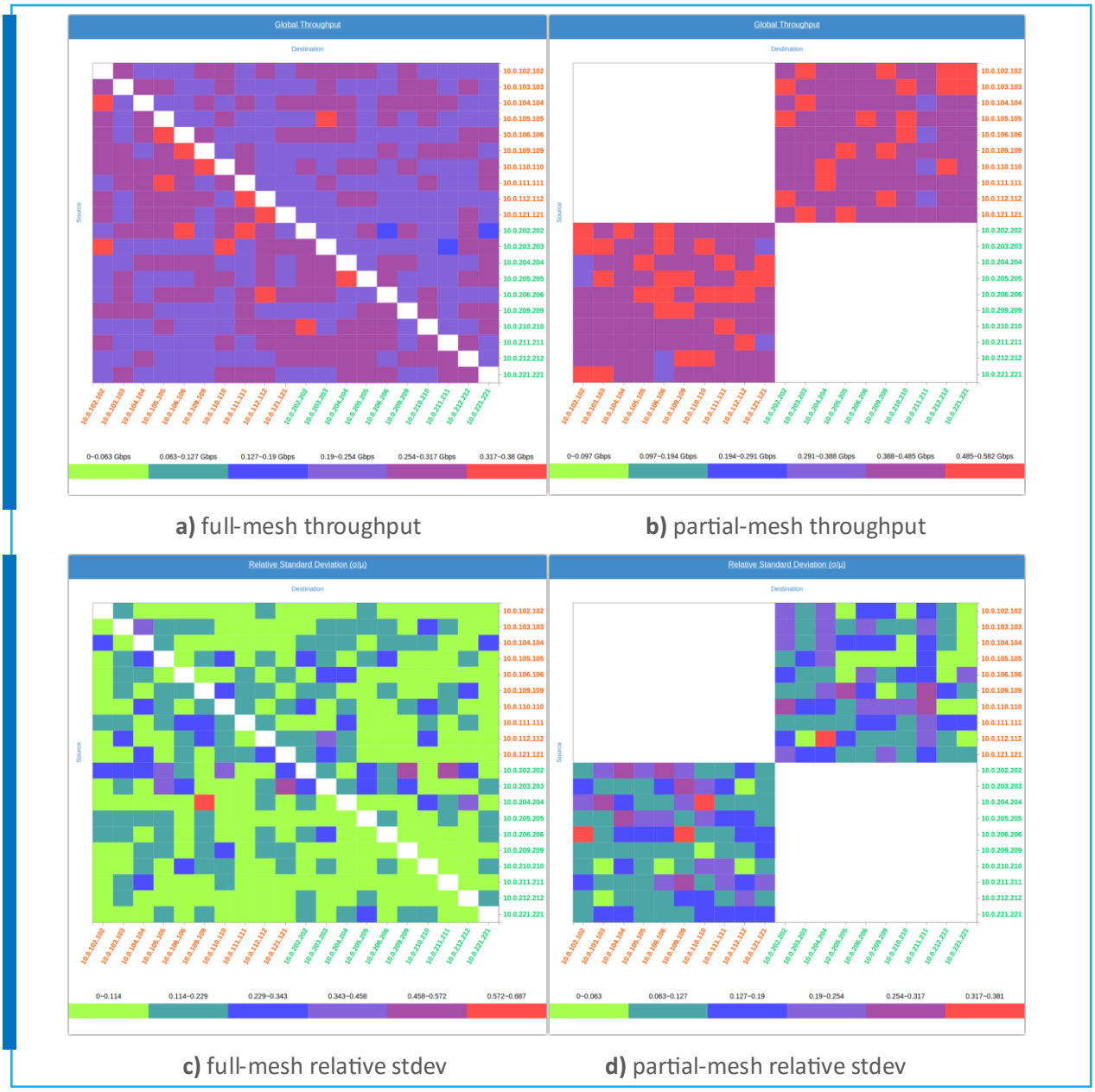
c) Flows bandwidth mean and standard deviation

Figure 2 Per node plots

PER PAIR HEAT-MAPS

Heat-maps provide the most detailed information for a given test, by presenting the statistics from the node-pairs prospective. Each heat-map has been created using six colour coded values ranges that are computed in real-time. Figure 3 illustrates some sample per-pair heat-maps.

- **Global Throughput**. While Figure 3a shows the per-pair throughput in a full mesh configuration, Figure 3b depicts the per-pair throughput in a partial mesh configuration. On the axis of the heat-maps, the two groups of nodes are highlighted with different colours. It is thus very easy to understand the traffic distribution by looking at the heat-maps.
- **Relative Standard Deviation**. The relative standard deviation is a standardized measure of dispersion of a probability distribution or frequency distribution and is defined as the ration between the standard deviation and the mean. The relative standard deviation heat-maps (see Figure 3c for a full mesh configuration and Figure 3d for a partial mesh one) is useful in order to understand how the bandwidth is shared between multiple TCP flows between every pair of nodes. The traffic distribution (e.g. full-mesh, partial-mesh) can be inferred from these heat-maps too.



a) full-mesh throughput **b)** partial-mesh throughput

c) full-mesh relative stdev **d)** partial-mesh relative stdev

Figure 3 Per pair heat-maps

4. CONCLUSION AND FUTURE WORKS

As a result of the work conducted during the project the NetBench software quality has been significantly improved in multiple areas, ranging from adding support for the iperf3 engine, and concentrating on the visualization plots and their generation speed. Furthermore, during the development several further possible improvements were identified:

- **UI improvements**. It would be better to use a continuous colour map in heat-map plots instead of a discrete one. Furthermore, plots such as *Flow difference* and *Flow Bandwidth MEAN and STD*, can be merged together in order to give a single plot overview of the running test.
- **Data storage improvements**. Instead using a standalone PostgreSQL, it can be envisaged to use the Database-On-Demand service offered by IT-DB of CERN, and thus safely preserve the test data.
- **Flow time-series**. All the existing plots give the latest snapshot of the test conditions. It is desired to be able to view the time-series of various flows generated in the test, in order to understand the stability of the test in time.

5. BIBLIOGRAPHY

[1] [RFC2544] - S. Bradner, "Benchmarking Methodology for Network Interconnect Devices," 1999.

[2] “iPerf & iPerf3,” ESnet / Lawrence Berkeley National Laboratory, [Online]. Available: https://iperf.fr/.